UNIVERSIDADE FEDERAL DE SANTA CATARINA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Arliones Stevert Hoeller Junior

Gerência do Consumo de Energia Dirigida pela Aplicação em Sistemas Embarcados

Dissertação submetida à Universidade Federal de Santa Catarina como parte dos requisitos para a obtenção do grau de Mestre em Ciência da Computação.

Prof. Dr. Antônio Augusto Medeiros Fröhlich

Florianópolis, Fevereiro de 2007

Gerência do Consumo de Energia Dirigida pela Aplicação em Sistemas Embarcados

Arliones Stevert Hoeller Junior

Esta Dissertação foi julgada adequada para a obtenção do título de Mestre em Ciência da Computação, área de concentração Computação Paralela e Distribuída e aprovada em sua forma final pelo Programa de Pós-Graduação em Ciência da Computação.

Prof. Dr. Rogério Cid Bastos

Banca Examinadora

Prof. Dr. Antônio Augusto Medeiros Fröhlich

Prof. Dr. Flávio Rech Wagner

Prof. Dr. Rômulo Silva de Oliveira

Prof. Dr. Mauro Roisenberg

Agradecimentos

Agradeço primeiramente a minha família pelo apoio incondicional durante meus estudos. Especialmente agradeço aos meus pais, Arliones e Elisabete, e meu irmão, Alexandre, pelo amor, carinho, atenção e compreensão, e por aceitar e entender um filho/irmão por muitas vezes ausente.

Gostaria também de agradecer aos meus amigos, muitos dos quais tenho como irmãos, que sempre estiveram dispostos a ajudar. A eles agradeço pelos momentos de diversão e alegria, certamente essenciais para manter-se motivado e centrado em qualquer momento da vida. Muito obrigado Beavis, Cabeça, Carlucci, Daniel, Gabriela, Ivan, Pelotas, Petrúcio, Rafa, Roberta, Pretinho, Zeh, e muitos outros, aos quais peço desculpas por não ter incluído o nome na lista.

Agradeço também aos meus colegas de laboratório (LISHA - Laboratório de Integração Software/Hardware) pelo companheirismo, auxílio e valorosas discussões sobre o meu trabalho. Em especial ao Lucas, Hugo e Fauze, que sempre estiveram próximos a mim e deveriam, sem dúvida alguma, estar citados no parágrafo anterior.

Agradeço ao Guto, meu orientador, pela orientação e ensinamentos, sem os quais a realização deste trabalho não seria possível. Obrigado pelo esforço despendido para tornar todas as pessoas que passam pelo LISHA ótimos pesquisadores. Graças a este esforço que foi possível pertencer a um grupo de pesquisa sério, bem sucedido e conceituado. Obrigado por ter orientado este trabalho e obrigado pela amizade.

Finalmente, agradeço não à Universidade Federal de Santa Catarina, mas sim a todos aqueles responsáveis por tornar esta instituição uma das mais bem conceituadas da América Latina. Agradeço especialmente à Verinha (Vera Lúcia Sodré Tei-

xeira), secretária do Programa de Pós-Graduação em Ciência da Computação, ao Prof. Dr. Raul Wazlawick, coordenador deste programa durante quase todo o tempo que estive vinculado a ele e ao Prof. Dr. Rogério Cid Bastos, atual coordenador do programa, pelo esforço em proporcionar aos seus alunos um curso de mestrado de alta qualidade.

À minha família.

Resumo

Baixo consumo de energia é um dos principais requisitos no projeto de sistemas embarcados, principalmente quando estes são alimentados por baterias. Técnicas que têm sido aplicadas com eficácia em sistemas de computação genérica não têm atingido o mesmo êxito em sistemas embarcados, ou devido à falta de flexibilidade, ou devido aos requisitos para sua implantação (volumes de memória e processamento), que podem tornar proibitiva sua aplicação nestes dispositivos.

Este trabalho define uma interface *simples* e *uniforme* para gerência de energia dirigida pela aplicação em sistemas embarcados. Esta interface disponibiliza ao programador da aplicação a flexibilidade de configurar os modos de operação de baixo consumo dos componentes em uso, conforme sua necessidade. A implementação buscou garantir a portabilidade desta aplicação a um baixo custo em termos de uso de memória e processamento. Este trabalho utiliza Redes de Petri Hierárquicas para especificar os procedimentos de troca de modos de operação dos componentes, utilizando os pontos de refinamento destas redes para representar as relações entre os diversos componentes do sistema. O uso das Redes de Petri permitiu analisar o mecanismo de gerência de energia para verificar seu funcionamento e a inexistência de impasses.

A extensão da interface dos componentes e a inclusão dos procedimentos de troca de modo de operação foram implementadas como um aspecto. Um protótipo foi desenvolvido utilizando o sistema operacional *Embedded Parallel Operating System* (EPOS) e estudos de caso foram realizados para demonstrar a usabilidade desta interface.

Abstract

Low power consumption is among the main requirements of embedded systems design, specially when these systems are battery-powered. Power management techniques that have been successfully applied on general purpose computing systems haven't achieved the same results on the embedded field. This happens either due to the lack of flexibility of such techniques or due to the resources (memory and processing) required to use them.

Within this context, this work defines a *simple* and *uniform* interface for application-driven power management of embedded system. This interface allows application programmers to configure low power operating modes of each component in order to satisfy their needs. Hierarchical Petri Nets are used to specify the operating mode transition procedures of components, using the refinement of these Petri nets to represent system components' interactions. The use of Petri Nets allowed the analysis of the proposed power management mechanism to verify its behavior and the absence of deadlocks.

The implementation of this mechanism focused on ensuring applications portability, while not incurring in unnecessary memory or processing overheads. The extension of the system components to implement the power management mechanism was done using aspect-oriented programming, thus not modifying original components. A prototype of the power manager was developed using the EMBEDDED PARALLEL OPERATING SYSTEM (EPOS) and case studies were performed to show the usability of the mechanism.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Siglas

ACPI Advanced Configuration and Power Interface

ADC Analog-to-Digital Converter

AML ACPI Machine Language

AOP Aspect Oriented Programming

AOSD Application Oriented System Design

API Application Programmer Interface

APM Advanced Power Management

CMAC Configurable MAC

CoS Class of Service

CPU Central Processing Unit

DPM Dynamic Power Management

DRAM Dynamic Random Access Memory

DVS Dynamic Voltage Scaling

eCos embedded Configurable operating system

EPOS Embedded Prallel Operating System

GPRS General Packet Radio Service

HAL Hardware Abstraction Layer

ISO International Organization for Standardization

MAC Media Access Control

NIC Network Interface Card

NoC Network-on-Chip

PNML Petri Net Modeling Language

PNTD Petri Net Type Definition

QoS Quality of Service

SPI Serial Peripheral Interface

SRAM Static Random Access Memory

TCP/IP Transmission Control Protocol / Internet Protocol

TLB Translation Lookaside Buffer

UART Universal Asynchronous Receiver/Transmitter

UML Unified Modeling Language

Capítulo 1

Introdução

Sistemas embarcados são plataformas computacionais utilizadas para monitorar e/ou controlar os espaços nos quais estão inseridos. Estes espaços podem ser máquinas, motores, dispositivos eletrônicos, ambientes físicos (e.g, módulos de sensoriamento em uma rede de sensores sem-fio monitorando um habitat), etc. Como grande parte deles são alimentados por baterias, é muito importante que estes sistemas sejam *power-aware*, i.e., capazes de gerenciar sua potência, possibilitando assim a diminuição do consumo de energia e o controle do aquecimento. Contudo, a maioria das metodologias, técnicas e padrões de software para este tipo de gerência não se mostram viáveis para sistemas embarcados que sofrem de severas limitações de recursos. Isto ocorre porque aquelas estratégias foram concebidas focando sistemas de propósito geral, onde custos adicionais de processamento ou memória são geralmente insignificantes.

ACPI [HP 04] e APM [INT 96] são os padrões mais utilizados pela indústria hoje para especificar a interface entre software e hardware no que diz respeito a controle do consumo de energia. Embora muito usados em dispositivos para sistemas de propósito geral, eles impõem requisitos de recursos adicionais de hardware ou capacidade de processamento que freqüentemente inviabilizam seu uso em sistemas embarcados. Componentes de sistemas embarcados apresentam uma grande variedade de características que podem ser configuradas para reduzir o consumo de energia. Padrões como estes (ACPI e APM) poderiam restringir esta configurabilidade. Além destes padrões,

diversas outras técnicas foram desenvolvidas para tratar o consumo de energia de sistemas eletrônicos. A maioria delas são sistemas dinâmicos que reúnem informação através da análise do comportamento do sistema, e usam esta informação para guiar as decisões acerca da gerência de energia (*Dynamic Power Management* - DPM) [BEN 98]. Exemplos de tais técnicas são heurísticas para aplicação de *Dynamic Voltage Scaling* (DVS), que, dinamicamente, ajustam a fonte de tensão e freqüência do processador para diminuir o consumo de energia.

Embora geralmente baseados em microcontroladores relativamente simples, dispositivos embarcados também permitem gerenciar energia provendo diferentes modos de operação e um grande conjunto de características configuráveis do hardware. Por exemplo, um microcontrolador ATMEGA [ATM 04a], da Atmel, oferece oito modos de operação diferentes e características configuráveis para quase todos seus componentes (e.g., ADC, UART, etc) que têm efeito direto no consumo de energia. Variações na tensão (e.g., operação em 3 V e 5 V) e na freqüência do processador também são possíveis com o auxílio de um circuito externo. Outros processadores como ARM, XSCALE e POWERPC possuem uma série muito mais complexa de modos de operação e recursos para gradação de tensão e freqüência. De fato, o hardware tipicamente usado em sistemas embarcados permite que o consumo de energia seja eficientemente gerenciado, mas os ambientes de software existentes (sistemas operacionais e bibliotecas para sistemas embarcados) não provém suporte adequado para este fim.

A maioria dos sistemas operacionais para plataformas embarcadas são compostas por simples camadas de abstração de hardware (*Hardware Abstraction Layers* - HAL) e oferecem pouco (ou nenhum) suporte de alto-nível para gerência de energia [POL 04]. Na maioria destes sistemas é esperado que as próprias aplicações implementem operações de controle de consumo de energia acessando a HAL. Dentre os problemas que surgem com estes mecanismos está o comprometimento da portabilidade e o aumento da complexidade da aplicação, já que o programador desta é forçado a adaptar seu código-fonte às peculiaridades do hardware que está utilizando.

Neste contexto surgem os sistemas operacionais baseados em componentes [SMA 95, FOR 97]. Estes sistemas buscam dividir o sistema operacional em arte-

fatos de software independentes (i.e., componentes) que implementam as funcionalidades desejadas (e.g., comunicação, processamento). Estes componentes podem ser utilizados para abstrair diferentes camadas do sistema como, por exemplo, utilizando um componente para implementar cada camada de uma pilha de protocolos de comunicação. Estas separações associadas a regras de composição de componentes têm permitido a geração de sistemas operacionais especializados para diferentes contextos, além de facilitar os processos de desenvolvimento e manutenção destes sistemas. Estes sistemas são atrativos do ponto de vista das aplicações ao passo que disponibilizam interfaces de programação (API) geralmente simplificadas e que abstraem um maior número de funcionalidades.

Sendo assim, este trabalho define uma interface de alto nível para gerência do consumo de energia de aplicações embarcadas, em sistemas baseados em componentes, que não implica em custos adicionais excessivos de processamento e memória enquanto mantém a portabilidade das aplicações e a facilidade no seu desenvolvimento. O foco principal deste trabalho está nos sistemas ditos *profundamente embarcados*, ou seja, sistemas computacionais embarcados, de pequeno porte e com funcionalidade específica, cujos requisitos de preço, tamanho e desempenho levam ao emprego de dispositivos de hardware extremamente simples, como microcontroladores que operam a baixas freqüências (muitas vezes na ordem de KHz) e que possuam pouca quantidade de memória (na ordem de KBytes).

Nesta proposta, a portabilidade é atingida através da definição de uma interface compacta e uniforme de gerência de energia para componentes de software e hardware. Além de métodos para o acesso pela aplicação, a interface é composta de uma estrutura que permite ao programador configurar os modos de operação que pretende utilizar em sua aplicação. Para facilitar o desenvolvimento das aplicações foi adotado um mecanismo para especificar as relações entre diferentes componentes do sistema. Este mecanismo é baseado em Redes de Petri Hierárquicas [PET 77], utilizando os pontos de especialização na hierarquia das redes para representar trocas de mensagens entre componentes do sistema. Através do mecanismo de trocas de mensagens é possível coordenar os componentes para que as trocas entre modos de operação sejam realizadas de forma correta. A facilitação no desenvolvimento das aplicações advém do fato de o programa-

dor apenas gerenciar os componentes que utiliza diretamente no sistema. Solicitando a troca de modo de operação destes componentes, o sistema fica responsável por propagar mensagens aos demais componentes do sistema.

1.1 Objetivos

O objetivo principal deste trabalho é explorar gerência de energia dirigida pela aplicação para permitir o controle do consumo de energia em sistemas profundamente embarcados sem implicar em adição excessiva de custos de processamento e consumo de memória e mantendo a portabilidade das aplicações. Para tanto, o sistema de gerência de energia proposto permite que a aplicação expresse quando certos componentes de software não estão sendo utilizados, fazendo com que o sistema troque estes componentes para estados de mais baixo consumo.

No desenvolvimento deste sistema foram identificados desafios a serem vencidos para permitir a transparência de vários aspectos relativos a, principalmente, diferenças arquiteturais entre dispositivos de hardware e à concorrência no acesso a recursos do sistema. São eles:

Definição de uma interface uniforme de gerência do consumo de energia: Uma interface foi especificada para permitir que a troca entre modos de operação dos diversos componentes do sistema possa ser realizada com a mesma semântica, contribuindo assim para a portabilidade. A interface definida é **simples**, i.e., possui um conjunto mínimo de métodos para gerência de energia, e **uniforme**, i.e., a mesma interface é apresentada por todos os componentes do sistema. Esta interface ainda apresenta um mecanismo de configuração dos modos de operação para cada componente, permitindo que o programador da aplicação selecione os modos de operação mais relevantes para sua aplicação.

Propagação de trocas de modo de operação: Em sistemas baseados em componentes é comum que programadores utilizem componentes de alto-nível, i.e, componentes que abstraem implementações complexas e que, muito freqüentemente, agregam

e utilizam outros componentes do sistema. Seria contraproducente exigir que o programador da aplicação conheça todos os componentes envolvidos na implementação destes componentes de alto-nível e gerencie o consumo de energia de todos eles individualmente. Para permitir uma gerência completa do consumo de energia foi necessário estabelecer um mecanismo pelo qual as trocas de modo de operação sejam propagadas através da hierarquia de componentes do sistema.

Especificação de um mecanismo de propagação das trocas de modos de operação: Foi identificada a necessidade de especificar o procedimento de troca de modos de operação dos componentes a fim de permitir que esta troca não gere inconsistências no sistema. Para tanto, Redes de Petri Hierárquicas foram utilizadas para descrever estes procedimentos. Para não agregar custos de processamento e memória em tempo de execução, uma análise destas redes é realizada em tempo de compilação, evitando a necessidade de um interpretador no sistema.

1.2 Estrutura do Texto

O capítulo 2 apresenta um levantamento do estudo realizado sobre gerência do consumo de energia em sistemas computacionais. Este estudo tenta deixar claro a grande variedade de técnicas desenvolvidas por pesquisadores e pela indústria para o controle de consumo de energia. O capítulo relata técnicas utilizadas para controlar o consumo de energia em diferentes níveis do projeto de um sistema computadorizado, i.e., do hardware à aplicação, ressaltando a necessidade da integração dos mecanismos utilizados nas diferentes camadas, através das técnicas denominadas *cross-layer adaptations*.

O capítulo 3 apresenta a proposta deste trabalho para gerência do consumo de energia em sistemas embarcados. Este capítulo descreve o mecanismo de especificação das trocas de modo de operação, a interface para gerência do consumo de energia em componentes de software e hardware e o mecanismo de propagação de mensagens de gerência de energia. O capítulo 4 descreve a implementação dos componentes da proposta deste trabalho no sistema operacional EPOS [FRö 01]. O capítulo 5 apresenta a análise

dos resultados obtidos em dois estudos de caso envolvendo implementações reais. O capítulo 6 finaliza o trabalho, sumarizando os resultados alcançados e discutindo futuras aplicações e extensões do sistema desenvolvido.

Capítulo 2

Gerência do Consumo de Energia

O avanço das tecnologias aplicadas na produção de computadores, embora permitam a produção de dispositivos muito mais rápidos e versáteis, fazem com que estes equipamentos passem a consumir muita energia. Conforme pode ser observado na figura 2.1, a densidade de potência[1] em dispositivos semicondutores cresce em escala exponencial conforme progride seu nível de integração. O aumento da densidade de potência nestes dispositivos traz à tona diversos problemas relativos à vida-útil dos *chips*, a alternativas de resfriamento e, de maior interesse neste trabalho, ao consumo de energia.

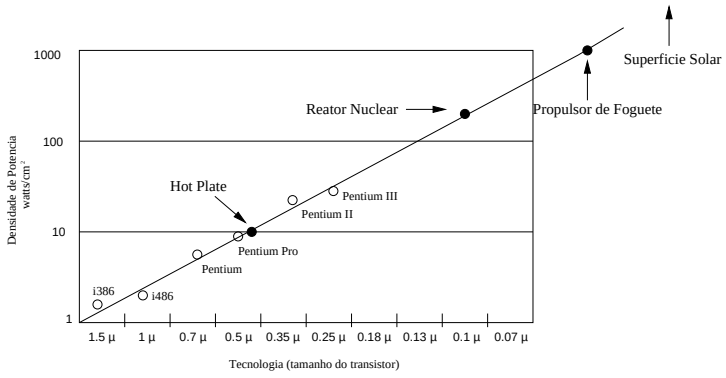


Figura 2.1: Progressão da densidade de potência em semicondutores (figura adaptada de [POL 99]).

[1]Densidade de potência é a quantidade de potência produzida por uma fonte de energia em uma determinada área ou volume.

Neste capítulo será, primeiramente, caracterizado o problema de consumo de energia em sistemas computacionais, apresentando como estes dispositivos consomem energia e em que pontos se pode agir para diminuir este consumo. Em seguida é apresentada uma revisão bibliográfica de técnicas de gerência de energia, incluindo técnicas de redução do consumo de energia desde o projeto do hardware até a geração de aplicações, porém com maior ênfase em técnicas que envolvam software, i.e., com participação do sistema operacional, compiladores ou aplicações.

2.1 Consumo de Energia em Sistemas Computacionais

> Potência e energia são comumente definidas em termos do trabalho realizado por um sistema. *Energia* é o total de trabalho que um sistema realiza em um período de tempo, enquanto *potência* é a taxa em que o sistema realiza este trabalho. [VEN 05]

Adaptando a definição acima ao contexto de computadores, o trabalho seria definido pela execução de programas, potência representaria a taxa em que este trabalho é realizado e energia seria o total de energia elétrica consumida ou dissipada como calor pelo sistema em um período de tempo. Neste contexto é importante diferenciar potência e energia porque, nem sempre, redução de potência implica em redução de energia. Embora a variação de uma afete a outra de forma linear (equação 2.1), algumas medidas que reduzem potência (P) podem afetar o tempo de execução (T) de um mesmo conjunto de tarefas em um processador, resultando em quantidades similares de energia total consumida (E). Por exemplo, reduzir a freqüência de um processador pela metade poderia, grosseiramente, reduzir a potência pela metade. Porém o sistema terá consumido a mesma quantidade de energia se essa redução na freqüência de operação dobrar o tempo necessário para realizar o mesmo conjunto de tarefas.

$$E = \int_{t_1}^{t_2} P \, dt \tag{2.1}$$

No contexto de sistemas embarcados, em especial os móveis, o impor-

tante é reduzir a energia total consumida pelo sistema, permitindo assim o aumento da vida-útil de suas baterias. Para permitir o controle deste quantidade de energia, técnicas tentam reduzir os dois tipos de potência existentes nestes dispositivos: potência dinâmica (*Dynamic Power*), que é decorrente da atividade do sistema, e potência estática (*Leakage Power*), que é a energia consumida mesmo quando um dispositivo está inativo.

Existem duas fontes de potência dinâmica: alterações nas cargas dos capacitores e curto-circuitos. Curto-circuitos, que ocorrem quando transistores adjacentes de diferentes polaridades trocam de estado simultaneamente, representam apenas 10% a 15% do total de potência dinâmica. Já as alterações nas cargas dos capacitores, que ocorre devido à atividade do circuito (carregando e descarregando os capacitores), é a fonte primária de potência dinâmica. Como além de não serem muito expressivos no cálculo da potência dinâmica total, os curto-circuitos são provenientes do projeto do hardware e, em muitos casos, inevitáveis, o cálculo da potência dinâmica é normalmente simplificado. Esta simplificação é demonstrada na equação 2.2 que apresenta a potência dinâmica como o cálculo das alterações nas cargas dos capacitores do circuito (P_{din}). Este valor é definido como sendo o produto da capacitância (C), do quadrado da tensão (V^2), da freqüência de operação do circuito (f) e do fator de atividade (a), que representa o número de transições lógicas entre 0 e 1 ou 1 e 0 que ocorrem em um *chip* [VEN 05].

$$P_{dinamica} \sim aCV^2f \tag{2.2}$$

Sucintamente, a potência estática dos circuitos semicondutores advém da imperfeição dos transistores. Transistores são dispositivos semicondutores com três terminais: gatilho (*gate*), fonte (*source*) e dreno (*drain*). O transistor opera controlando o fluxo de corrente entre seus terminais *fonte* e *dreno* baseado na tensão aplicada em seu terminal *gatilho*. Em situação ideal, estes transistores deveriam operar liberando o fluxo de corrente apenas quando a tensão aplicada no gatilho atingisse um determinado limite. Contudo, os transistores permitem que uma determinada quantidade de corrente flua entre os terminais fonte e dreno mesmo quando a tensão aplicada está abaixo do limite desejado. Esta corrente, conhecida como corrente de fuga, é responsável pela potência

estática, conforme demonstra a equação 2.3. Este tipo de potência está se tornando a fonte dominante da potência total em circuitos [POL 99].

$$P_{estatica} = VI_{fuga} \tag{2.3}$$

Retomando o exemplo anterior da diminuição da freqüência de operação de um processador, agora fica claro que simplesmente reduzir pela metade a freqüência de operação de um circuito, na verdade, pode aumentar o consumo de energia deste. Isso ocorre porque, além de não alterar o consumo de energia devido à potência dinâmica, este dispositivo ficaria por mais tempo consumindo energia devido à potência estática. Sendo assim, tentativas de reduzir a potência destes circuitos têm que abordar, além da freqüência, outras grandezas que influem no consumo de energia, como a tensão de operação, que implica em diminuição de ambos os tipos de potência, e a capacitância dos circuitos, que reduz a potência dinâmica.

2.2 Técnicas de Redução do Consumo de Energia no Projeto de Hardware

Há várias medidas que são utilizadas para diminuir potência no projeto de hardware. Estas medidas vão desde a concepção dos transistores e circuitos até a especificação das hierarquias de memória. Uma técnica bastante utilizada no processo de concepção de transistores é a alteração de tamanho (*sizing*). Quanto menores os transistores, menos potência dinâmica eles consomem. Esta diminuição, contudo, aumenta o atraso na propagação de sinais. Sendo assim, pesquisadores têm trabalhado em algoritmos para determinar pontos críticos em circuitos, definindo atrasos aceitáveis para cada ponto, permitindo estimar o tamanho ideal dos transistores de modo a não violar estes atrasos [PEN 02, EBE 04]. Técnicas de reordenação procuram identificar transistores que trocam de estado muito freqüentemente e posiciona-los próximos às saídas dos circuitos, evitando que suas trocas de estados afetem outros transistores e, em um efeito dominó, aumente ainda mais a potência [KUR 04, SUL 04]. Outras técnicas utilizadas para reduzir

potência em circuitos são *Half Frequency* e *Half Swing Clocks*. *Half Frequency* ajusta os circuitos para que seus eventos ocorram tanto na borda de subida quanto na de descida do *clock*, permitindo a redução da freqüência de operação a metade. Técnicas de *Half Swing Clocks* usam sinais de *clock* de baixa tensão, o que permite uma redução quadrática da potência dinâmica (equação 2.2).

Vários trabalhos exploram técnicas para economizar energia no nível de projeto de circuitos lógicos. Venkatachalam descreve métodos de organização de portas lógicas mais eficientes em termos de potência [VEN 05]. Como exemplo, ele cita o caso da implementação de um circuito lógico AND com quatro entradas. Ele analisa os efeitos de implementar este circuito encadeando portas AND de 2 entradas (figura 2.2a) ou organizando estas portas em uma estrutura de árvore (figura 2.2b). Neste exemplo, em circuitos onde todas as entradas (A, B, C, D) possuem as mesmas probabilidades de adquirir valor lógico 1 ou 0, uma estrutura em cadeia (figura 2.2a) seria mais indicada, pois apresentaria menos trocas de estados nas saídas das portas lógicas, diminuindo o fator de atividade e, por conseqüência, a potência dinâmica (equação 2.2). Esta topologia pode, contudo, gerar atrasos na propagação dos sinais e, eventualmente, valores incorretos na saída, sendo necessário a inserção de flip-flops ou registradores para sincronizar os sinais. Devido ao grande número de possibilidades de implementação de transistores e organização de circuitos lógicos, pesquisadores têm trabalhado para desenvolver algoritmos capazes de mapear modelos de circuitos definidos no nível de portas lógicas em diferentes estruturas de portas ou diferentes tecnologias de implementação [CHE 04, LI 04, RUT 01]. Outros pontos pesquisados no âmbito do projeto de hardware de baixa potência são o desenvolvimento de flip-flops e lógica de controle de menor potência.

Além dos circuitos, os mecanismos de interconexão afetam a potência, já que é através dos meios de interconexão que se dá a maior parte da atividade elétrica. Técnicas para evitar o aumento do consumo de energia em mecanismos de interconexão envolvem *Bus Encoding*, que busca diminuir as trocas de estado nos fios destes barramentos, e inserção de *shield wires* para evitar interferências entre fios (*Crosstalk*). Outras técnicas tentam eliminar os mecanismos de codificação, permitindo maiores trocas de estado nos fios, porém operando em baixas tensões, diminuindo a potência do barramento.

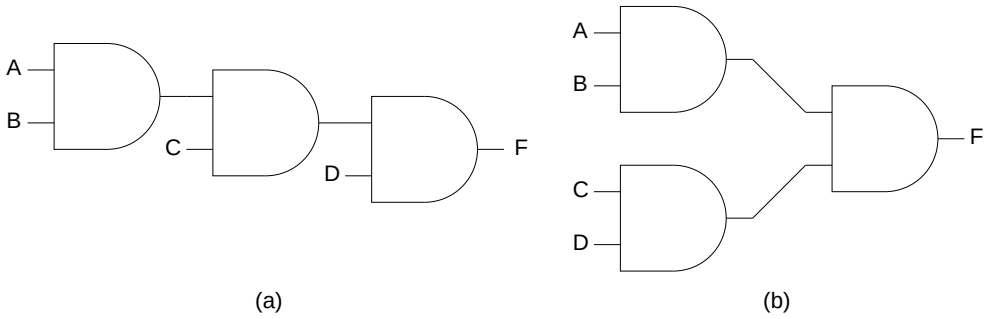


Figura 2.2: Reestruturação de portas lógicas. Adaptada de [VEN 05]

Também são adotadas técnicas de segmentação de barramentos que permitem que os sinais atinjam somente os setores do barramento onde são necessários. Todas estas técnicas prevêem a existência de uma arquitetura em que várias unidades funcionais compartilhem um ou mais barramentos. A fim de tratar problemas como atraso (devido a concorrência e *handshaking*) e potência existentes neste modelo, pesquisadores têm desenvolvido redes intra-chip (*Network-on-Chip* - NOC). Estas técnicas permitem aplicar mecanismos de redução de atraso e controle de tráfego à interconexão de unidades funcionais dentro de um *chip*, impactando em melhoria do desempenho e redução do consumo de energia.

As memórias também contribuem para o consumo de energia em sistemas. Algumas técnicas para reduzir este consumo focam na divisão das memórias em subsistemas menores, permitindo que bancos de memória troquem seus modos de operação para estados de menor consumo de energia separadamente. Divisões de baixa granularidade associadas a técnicas de software que aumentam a localidade espacial de dados permitem concentrar o uso da memória a um pequeno conjunto dos bancos definidos, permitindo que os demais bancos entrem em modos de operação de menor potência. Aumentando a granularidade desta divisão, cada um dos bancos de memória poderiam ser divididos em “sub-bancos”, permitindo ativar apenas os sub-bancos relevantes em cada acesso à memória. Outras técnicas exploram diferentes hierarquias de memória e estratégias para a gerência destas hierarquias. Devido a questões como desempenho e custo, diferentes tecnologias são utilizadas em diferentes níveis da hierarquia de memória, incluindo SRAMs (*Static Random Access Memories*) para caches, DRAM (*Dynamic*

Random Access Memories) para memória principal e, em algumas situações, discos rígidos para *swap*, entre outras. Basicamente, quanto mais alto o nível do acesso à memória, maior o atraso e a quantidade de energia consumida. Portanto, técnicas que diminuem a quantidade de faltas nas caches e nas TLBs (*Translation Lookaside Buffers*) dos processadores contribuem para um menor consumo de energia.

2.3 *Dynamic Voltage Scaling* - DVS

Uma das técnicas mais eficientes para reduzir o consumo de energia em sistemas computacionais é a redução da tensão de operação destes circuitos. Porém, como a diminuição do nível de tensão acarreta atrasos na propagação de sinais (*gate delays*), é necessário reduzir também a freqüência de operação para manter a sincronia do circuito. O processo de conciliar variações de tensão e freqüência é chamado de *Dynamic Voltage Scaling* (DVS) [CHA 92]. A diminuição da tensão de operação é bastante eficiente, pois afeta linearmente a potência estática (equação 2.3) e quadraticamente a potência dinâmica (equação 2.2), sendo largamente utilizada, principalmente em processadores. Como DVS tem um efeito direto sobre o desempenho, trabalhos nesta área têm desenvolvido heurísticas que exploram trocas entre consumo de energia e desempenho com base no comportamento do sistema e/ou aplicações. A aplicação destas heurísticas é, contudo, complexa devido à natureza imprevisível dos *workloads*, além do indeterminismo e das anomalias apresentadas por sistemas reais. Tradicionalmente, as técnicas DVS são classificadas em *baseadas em intervalos*, *inter-processos* ou *intra-processos*.

Técnicas baseadas em intervalos de tempo monitoram o comportamento do sistema e, conhecendo a taxa de utilização do dispositivo no último intervalo, estimam a taxa de utilização para o próximo ajustando a velocidade e a tensão. O que diferencia as técnicas desta categoria é o modo de estimar a taxa futura de utilização. Um dos primeiros algoritmos baseados em intervalos foi o PAST [WEI 94]. O PAST mede periodicamente o tempo de ociosidade do processador, diminuindo sua velocidade sempre que este tempo ultrapassa um limite. De modo similar, sempre que o processador permanece ocupado por um tempo maior que um limite, a velocidade é aumentada. Como o PAST baseia suas

decisões apenas na janela de tempo mais recente ele está muito susceptível a erros. Várias extensões foram feitas a este algoritmo para aumentar sua eficiência. Grande parte destas modificações permitiram basear sua decisão em uma maior quantidade de informação, aumentando a janela de tempo analisada [GOV 95]. Contudo, todos estes algoritmos assumem que os *workloads* são regulares, o que raramente é verdade.

Técnicas inter-tarefa (*intertask*) definem diferentes modos de operação para cada tarefa. Assim, o escalonador do sistema, ao realizar a troca de contexto, configura os dispositivos de hardware com base no comportamento passado da aplicação a ser carregada. Neste contexto, Weissel e Bellosa propuseram o *Process Cruise Control* [WEI 02a]. Esta técnica utiliza contadores de eventos, presentes em processadores modernos, para monitorar a ocorrência dos eventos que impactam no consumo de energia (e.g., períodos de computação intensiva, acessos a memória, uso de barramentos). A técnica ajusta a configuração do hardware consultando uma tabela que relaciona diferentes comportamentos (obtidos através de simulação) com a freqüência mais baixa em que a tarefa possa executar sem comprometer o nível de desempenho desejado. Uma das desvantagens destas técnicas está relacionada ao fato de que, normalmente, o *workload* das tarefas é desconhecido. Isso implica em conhecimento profundo das tarefas (e.g., utilizando simulações para o pior caso) ou a basear as decisões utilizando *workloads* anteriores. Se os *workloads* forem irregulares, estimá-los com eficácia é difícil. Alguns trabalhos tentam minimizar este problema classificando os *workloads* e utilizando diferentes heurísticas para cada classe [FLA 01].

Técnicas intra-tarefa (*intratask*) gerenciam o consumo de energia durante a execução das tarefas. Uma das técnicas intra-tarefa é *run-time voltage hopping* [LEE 00]. Essa técnica divide cada tarefa em fatias de tempo de tamanho fixo e, para cada fatia de tempo, um algoritmo atribui a velocidade mais baixa que permite à tarefa executar dentro do tempo desejado. Esse algoritmo, contudo, é pessimista, já que utiliza simulações baseadas no pior caso de execução para calcular as velocidades em que a tarefa irá executar, fazendo com que o tempo de execução seja sempre muito parecido, senão igual, ao obtido na simulação para o pior caso. Há variações desta técnica, entre elas cabe citar PACE [LOR 01] e *Stochastic* DVS [GRU 01]. Estas duas técnicas utilizam

modelos probabilísticos para estimar o *workload* do ciclo de execução de uma tarefa baseado nos ciclos anteriores. Contudo, elas utilizam modelos simplificados do consumo de energia que podem não corresponder ao consumo de energia em sistemas reais.

As três técnicas intra-tarefa citadas acima são implementadas no nível do sistema operacional. Contudo, há outras técnicas desta categoria que são implementadas ao nível de compiladores. Uma destas técnicas realiza *profiling* de aplicações para identificar diferentes fluxos e tempos de execução, inserindo instruções para alterar a velocidade de operação do processador no início de cada fluxo, fazendo com que as tarefas consumam menos energia quando não executam segundo o pior caso [SHI 01]. *Program checkpointing* [AZE 02] marca tarefas com pontos de verificação e define tempos de execução para os trechos da tarefa entre estes pontos. Através de *profiling* ele determina o número médio ciclos entre cada ponto, inserindo código para ajustar a velocidade do processador em cada ponto de verificação. Outra técnica [HSU 03] realiza *profiling* das aplicações considerando todas as possibilidades de freqüência de operação em diferentes regiões de um programa, construindo uma tabela que relaciona a freqüência com o impacto no tempo de execução e no consumo de energia para cada região. A partir desta tabela, são selecionadas as combinações de regiões e freqüências que economizam mais energia sem ultrapassar um limite de tempo para execução.

2.4 Hibernação de Recursos

Como ressaltado na seção 2.1, componentes de sistemas computadorizados consomem energia mesmo quando estão ociosos devido à potência estática. Para reduzir o consumo de energia nestes casos são empregadas técnicas de hibernação de recursos. Estas técnicas implementam heurísticas que “desligam” os dispositivos em períodos ociosos. Este “desligamento” normalmente utiliza modos de operação que inibem a passagem de corrente pelo circuito do dispositivo. O impacto da hibernação temporária é normalmente bastante expressivo no consumo de energia, porém há alguns cuidados que devem ser tomados. Além da detecção dos períodos de ociosidade, as heurísticas têm que levar em consideração dois fatores: muitos componentes consomem uma quantidade

maior de energia para desligar e religar; e o acesso a componentes desligados acarreta atrasos que podem impactar significativamente no desempenho do sistema.

O processo de hibernação e reinicialização do processador de um sistema normalmente é muito custoso em termos de processamento e consumo de energia, sendo esta técnica utilizada apenas quando existe a certeza de que o sistema permanecerá hibernado por um período muito longo. Contudo, a hibernação de periféricos é um recurso bastante utilizado e eficaz. Sendo assim, técnicas de hibernação de recursos têm focado principalmente em dois tipos de dispositivos: discos e placas de rede (com ou sem fio). Técnicas de gerência de disco implementadas em sistemas operacionais tradicionais procuram reduzir a velocidade ou parar a rotação quando o tempo de ociosidade do disco atinge um determinado limite. Técnicas mais avançadas utilizam heurísticas que permitem ajustar este limite de tempo conforme o comportamento do sistema e das aplicações. Gerência do consumo de energia em placas de rede apresentam um maior desafio para os mecanismos de hibernação. Este desafio é o fato de que o simples desligamento de uma placa de rede pode desconectar o dispositivo de servidores ou outros *hosts* aos quais está conectado. Portanto, as técnicas para hibernar estes dispositivos envolvem o desenvolvimento de protocolos para sincronizá-los com os demais dispositivos conectados a sua rede.

2.5 Estratégias Para Gerência de Energia

Gerência de energia dirigida pelo sistema operadional (*OS-directed power management*) está associada às técnicas que utilizam o sistema operacional como entidade gerenciadora de energia. Os mecanismos de gerência de energia que adotam esta metodologia, o que inclui as implementações da maioria das técnicas discutidas nas seções 2.3 e 2.4, o fazem por considerar que o sistema operacional é o componente que possui a informação necessária para realizar essa gerência, ou seja, conhecimento relativo ao comportamento das aplicações e acesso para monitorar e configurar o hardware.

Gerência de energia dirigida pelo sistema operacional é utilizada pela grande maioria dos sistemas operacionais, tanto para computação genérica (e.g., WIN-

DOWS, LINUX), quanto para computação dedicada/embarcada (e.g., μCLINUX, VXWORKS, ECOS). Nestes sistemas os gerenciadores de energia são implementados ou como extensões dos escalonadores (para gerência da CPU) ou dentro dos drivers de dispositivos (para gerência de periféricos como discos ou placas de rede). A separação das políticas em diferentes partes do sistema torna complexa a tarefa de manter e integrar a gerência de energia, ou seja, implementar uma entidade única que seja capaz de gerenciar todos os componentes do sistema de modo uniforme.

Além das tecnologias de gerência de energia controladas por software, existem alguns recursos de gerência de energia que são implementados pelo hardware de forma transparente ao software, principalmente por processadores. A grande maioria dos processadores que suportam DVS configuram sua tensão de operação com base em uma tabela que contém níveis de tensão para operação em diferentes freqüências. Esta tabela é construída através de uma análise prévia de pior caso, garantindo que os níveis de tensão utilizados sejam seguros para a operação do processador. Processadores IA32 recentes da INTEL implementam DVS através da tecnologia SPEEDSTEP, que permite utilizar freqüências de operação variando a cada 100 MHz a partir de 40% da freqüência máxima do processador. Além desta tecnologia, modelos mais recentes como PENTIUM M e CENTRINO também implementam várias técnicas que reduzem a atividade de regiões do processador que não estão em uso. A AMD possui uma tecnologia semelhante chamada POWER NOW!. A tecnologia LONGRUN, empregada nos processadores CRUSOE e EFFICEON da TRANSMETA, também implementa ajustes de tensão de operação baseados em tabelas. Porém, a versão mais recente do LONGRUN atribui diferentes tensões para diferentes partes do circuito, economizando mais energia.

Embora nestes exemplos o ajuste da tensão seja feito automaticamente pelos processadores, continua existindo uma participação importante do software neste processo, já que a tensão só é ajustada a partir do momento que o software solicita um ajuste da freqüência, ou seja, a tomada de decisão continua vindo do software, principalmente do sistema operacional. Outra desvantagem destes sistemas é o uso de níveis de tensão obtidos através de análise de pior caso. Para melhorar a eficiência do DVS em processadores, a ARM desenvolveu um mecanismo de ajuste automático de tensão,

o RAZOR PIPELINE [ERN 03]. Este mecanismo utiliza a detecção de atrasos no circuito que ocorrem quando o nível de tensão está muito baixo para operação numa determinada freqüência. O RAZOR inclui hardware adicional para monitorar e corrigir estes atrasos. Uma política monitora a ocorrência dos atrasos, aumentando a tensão de operação quando muitos atrasos estão ocorrendo, e diminuindo a tensão enquanto a quantidade de atrasos estiver num limite aceitável. Como resultado, os pesquisadores obtiveram apenas 3% de redução no desempenho do sistema (devido aos atrasos corrigidos), enquanto conseguiram economizar até 64,2% da energia consumida.

Alguns sistemas operacionais embarcados, tais como TINYOS e ECOS, exportam interfaces que permitem às aplicações gerenciar o consumo de energia de componentes do sistema. Estas interfaces, contudo, são muitas vezes incompletas, limitando a configuração dos dispositivos, ou não padronizadas, comprometendo a facilidade de implementação e portabilidade da aplicação. Por exemplo, o TinyOS, um sistema operacional desenvolvido na Universidade de Berkley para sistemas de sensoriamento [HIL 00], possui uma interface de controle de dispositivos chamada `StdControl` que permite às aplicações ligá-los (`StdControl.start()`) ou desligá-los (`StdControl.stop()`). Neste sistema, para que a aplicação coloque um dispositivo em algum modo alternativo de baixo consumo de energia, precisará utilizar outros métodos da interface do componente ou implementar ela mesma os procedimentos de migração entre modos de operação.

2.6 Adaptações Multi-Camada

Para desenvolver um sistema que consuma energia de forma eficiente é necessário dar atenção a aspectos relacionados a todas as camadas que formam este sistema, desde os transistores até as aplicações. Por isso, uma questão que tem sido cada vez mais freqüente em trabalhos científicos da área é como desenvolver modelos abrangentes, que integrem informações presentes em vários níveis (e.g., sistema operacional, compilador e aplicação) nas decisões de gerência de energia. Para tentar responder a esta pergunta, alguns sistemas têm sido construídos para permitir as chamadas *cross-layer adaptations*, com destaque para os sistemas Forge [MOH 05] e Grace [SAC 06].

O Forge é um framework de gerência de energia para aplicações multimídia. Este sistema foca um cenário típico em que usuários com dispositivos portáteis (e.g., PDA, celular) requisitam vídeos pela rede. O sistema Forge filtra estas requisições através de um *proxy*, transcodificando e transmitindo os vídeos para cada cliente conforme o nível de qualidade de serviço (*Quality of Service* - QoS) mais eficiente em termos de energia. O Forge integra várias camadas em seu framework de gerência de energia. No nível do hardware, ele provê interfaces para DVS e para a placa de rede, permitindo desligá-la quando ociosa. Acima do hardware estão o sistema operacional e o compilador, que controlam as configurações do hardware, e acima do sistema operacional, está um middleware distribuído, que possui uma parte no dispositivo móvel e outra parte no proxy. O middleware em cada dispositivo móvel monitora as estatísticas relevantes ao consumo de energia e repassa esta informação ao middleware no proxy. De posse desta informação, o middleware no proxy ajusta o tráfego na rede e decide em qual nível de QoS deve transmitir cada vídeo requisitado.

O Grace é outro framework que busca adaptar várias camadas de gerência de energia. Ele tenta integrar DVS, escalonamento eficiente em termos de energia e configurações de QoS. Ele foi concebido para suportar aplicações multimídia de tempo real com períodos e prazos fixos. Este sistema inclui dois níveis de adaptação: global e local. Adaptações globais são realizadas por um coordenador central que monitora o nível da bateria e a demanda de processamento do sistema, atuando apenas quando estes sofrem grandes variações. Adaptações locais, contudo, respondem a pequenas variações dos workloads em cada tarefa. Estas adaptações são realizadas por três "adaptadores locais", um para configurar a freqüência da CPU (DVS), um para atuar no escalonamento de tarefas e um terceiro para adaptar os parâmetros de QoS. Alterações importantes no estado do sistema (e.g., diminuição drástica do nível da bateria, aumento muito grande da demanda por processamento) acionam o coordenador central para que este dispare um novo conjunto de adaptações globais. Estas adaptações, que são propagadas para os adaptadores locais, podem ser, por exemplo, solicitações para que os serviços não críticos sejam desativados, ou que tarefas classificadas em uma classe de serviço (*Class of Service* - CoS) inferior sejam suspensas. As ações que executam as decisões tomadas pelo coordenador global

são implementadas pelos adaptadores locais, que são livres para adaptar estas diretrizes globais conforme a execução de cada tarefa.

Outro trabalho explora como compilador e sistema operacional podem interagir para economizar energia [ABO 03]. Esse trabalho realiza uma análise de aplicações tempo real com prazos fixos para obter, através de simulação, os tempos de execução de pior caso para vários fluxos de execução diferentes na aplicação analisada. O compilador utiliza estes tempos de execução para manter em um registrador o número de ciclos restantes para que a tarefa finalize segundo o pior caso. O sistema operacional acessa periodicamente este registrador, ajustando a velocidade do processador de modo a garantir que a tarefa sempre finalize dentro do tempo previsto. Técnicas semelhantes a estas já foram descritas na seção 2.3, porém neste método, não são trechos de código inseridos pelo compilador que alteram a configuração do processador, quem configura o hardware é o sistema operacional. Isso permite à política de gerência de energia implementada pelo sistema simplesmente atender o prazo da tarefa em questão ou, devido à existência de outros eventos mais prioritários (e.g., bateria em nível muito baixo), permitir que a tarefa perca o prazo.

2.7 Interfaces e Padrões

Tentativas de padronização de interfaces para gerência de energia tiveram origem em sistemas de propósito geral. Antes do surgimento das primeiras tentativas de padronização, as funcionalidades de gerência de energia estavam limitadas a procedimentos no firmware de computadores que implementavam desligamento dos monitores e mecanismos de hibernação do sistema para a memória ou disco. Contudo, estes procedimentos eram realizados pelo firmware sem intervenção ou sequer conhecimento do sistema operacional. Outro fator complicante das técnicas até então era o fato de que cada fabricante implementava as funcionalidades de modo diferente. Neste contexto, o APM (*Advanced Power Management*) [INT 96] surgiu para unificar as funcionalidades de gerência de energia, e permitir ao firmware expor ao sistema operacional uma interface de gerência de energia. Isto permitiu ao sistema operacional ter informações importantes

sobre o sistema como o nível da bateria, além de controlar alguns aspectos de gerência de energia, como requisitar a hibernação do sistema para disco ou memória. As ações efetivas de gerência de energia, contudo, continuavam sendo realizadas pelo software contido no firmware, apresentando uma série de desvantagens [GRO 03]:

- falta de controle oferecido ao sistema operacional sobre os aspectos de gerência de energia (por exemplo, o mecanismo de hibernação do sistema era executado totalmente pelo firmware, não permitindo ao sistema operacional adaptá-lo ao seu estado atual);

- falta de flexibilidade da interface (e.g., o indicador de estado da bateria agrega informação, possivelmente de múltiplas baterias, em um simples valor que indica "minutos de bateria restantes");

- a falta de extensões da interface para uma maior variedade de dispositivos (e.g., placas de rede, discos);

- nos computadores baseados na arquitetura INTEL X86, os procedimentos do APM precisam ser executados com o processador operando em Modo Real, o que se tornou um inconveniente ao passo que os sistemas começaram a operar em modo protegido, além do fato de que, a partir do momento que um procedimento no firmware está sendo executado, o sistema operacional não está mais no controle do sistema. Uma chamada de procedimento do firmware poderia tomar muito tempo para retornar, ou fazer algo inesperado, ou, até mesmo, nunca retornar.

Para substituir o APM foi proposto o ACPI (*Advanced Configuration & Power Interface*) [HP 04]: uma interface mais flexível, abrangente e com extensões que eliminam a necessidade de executar código estranho ao sistema operacional, como ocorria com os procedimentos APM contidos no firmware. O ACPI identifica o sistema operacional como a entidade que tem conhecimento abrangente sobre os componentes de hardware e seu uso, além de também conhecer as características e comportamento das aplicações que acessam estes componentes. Em contraste ao APM, o sistema operacional

tem total controle sobre os modos de operação e peculiaridades da gerência de energia do hardware. ACPI foi projetada para não realizar chamadas diretas ao firmware. Ao invés de armazenar procedimentos de gerência de energia em código nativo pronto para ser executado, o firmware com suporte a ACPI passa a armazenar estes procedimentos em uma linguagem chamada AML (*ACPI Machine Language*). Este código é então executado pelo sistema operacional através de uma máquina virtual. A interpretação do AML trás algumas vantagens importantes:

- permite ao sistema operacional evitar a execução de código errôneo ou malicioso;
- a linguagem de máquina do ACPI abstrai tanto o sistema operacional, quanto a arquitetura ou plataforma em que está executando, retirando dos fabricantes da obrigação de fornecer suporte a drivers para vários sistemas ou arquiteturas diferentes.

Contudo, ao se considerar a adequação desta interface para sistemas embarcados, são identificados três maiores inconvenientes:

- o ACPI abstrai os modos de operação do hardware de um modo que pode ser muito restritivo para sistemas embarcados. Usando como exemplo um módulo de sensoriamento MICA2 MOTE [HIL 00], é observado que pode ser importante suportar todos os modos de baixo consumo de energia ou de maior desempenho do dispositivo de comunicação (rádio) para minimizar o consumo de energia de forma eficiente. O ACPI permite a definição de, no máximo, quatro modos de operação para cada dispositivo (D0 - D3), o que pode ser de granularidade muito grossa para aplicações embarcadas.
- o uso de uma linguagem interpretada para acessar os componentes de hardware, embora apresente vantagens substanciais, impõe requisitos ao sistema que podem exceder, em muito, os recursos limitados da maioria dos dispositivos embarcados. Nestes sistemas, quantidade de memória disponível ou espaço para código podem ser da ordem de poucos quilobytes, requerendo a existência de uma interface que permita uma implementação com tamanho de código minimizado e agregando pouco custo de memória e processamento.

- o papel do ACPI (assim como era do APM) é abstrair a interface entre sistema operacional e dispositivos de hardware. Sendo assim, mesmo melhorando consideravelmente o padrão anterior, o ACPI ainda mantém a padronização num nível baixo de abstração. O modo como o sistema operacional realiza a interface entre sua camada de abstração de hardware e seus componentes de mais alto nível, assim como o modo pelo qual funcionalidades de gerência de energia são exportadas para aplicações, não fazem parte deste padrão. Isso implica em interfaces diferentes entre sistemas operacionais ou na ausência de padrão até mesmo entre componentes do mesmo sistema. Alguns sistemas sequer implementam algum tipo de interface de alto nível de gerência de energia. Esta grande variedade de situações impede que aplicações sejam desenvolvidas eficientemente levando em conta critérios de gerência de energia.

2.8 O Papel da Aplicação na Gerência do Consumo de Energia

Vários trabalhos têm sido realizados para dar às aplicações um papel mais ativo nas decisões de gerência de energia. Ellis [ELL 99] realizou uma primeira análise que constatou os benefícios do envolvimento das aplicações na gerência de energia, defendendo a criação de uma API que permita o envolvimento das aplicações de forma ativa no processo de gerência de energia. Seguindo esta mesma visão, outros trabalhos realizados neste sentido focaram em três principais pontos: transformações nas aplicações, aplicações adaptativas e concepção de interfaces que permitam às aplicações influenciar o gerenciador de energia ativamente.

Tan et al. [TAN 03] propõem uma metodologia para projeto de aplicações de baixo consumo de energia. Neste trabalho os autores analisam a estrutura das aplicações e artefatos do sistema por ela utilizados (processos/*threads*, tratadores de eventos, *device drivers*), bem como a comunicação entre estes artefatos (sincronização e mecanismos de comunicação) para montar um grafo que represente o comportamento da

aplicação e a interação com os artefatos envolvidos. A partir desta representação de alto nível da aplicação são realizadas simulações para estimar o consumo de energia da aplicação original. Em seguida, transformações são aplicadas ao grafo original a fim de reduzir o consumo de energia. Estas transformações envolvem fusão de processos para reduzir a comunicação inter-processos (IPC), escolha de mecanismos de IPC mais eficientes e migração de trechos de computação entre processos.

Outros trabalhos exploram o desenvolvimento de aplicações adaptativas que permitam trocas entre exatidão das tarefas e consumo de energia. Sachs et al. propõe um codificador de vídeo que permite variar a eficiência da compressão não realizando algumas fases deste processo (*Motion Search* e DCT - *Discrete Cosine Transform*) [SAC 03]. No protótipo implementado, o codificador executou sobre um processador com suporte a DVS e a adaptações do hardware (e.g., caches configuráveis). O gerente de energia consiste de duas tarefas que operavam em conjunto. Uma tarefa configura os parâmetros do hardware no início da codificação de cada frame de vídeo, enquanto outra ajusta os parâmetros do codificador enquanto o frame está sendo processado.

Trabalhos explorando trocas entre QoS das aplicações e consumo de energia também foram realizados. Um sistema operacional que realiza este tipo de gerência de energia é o Odyssey [FLI 99]. O Odyssey implementa um framework para aplicações multimídia e Web. O sistema monitora os recursos utilizados pelas aplicações e envia um alerta às aplicações cujos recursos estão se tornado escassos, exigindo que a aplicação diminua sua qualidade de serviço até que os recursos estejam disponíveis novamente. Por exemplo, uma aplicação de visualização de mapas ao receber um alerta de baixo nível de bateria poderia passar a solicitar imagens de menor qualidade pela rede, cujos arquivos seriam menores, diminuindo o fluxo de dados e, conseqüentemente, diminuindo o consumo de energia. Outro sistema operacional que suporta a adaptação das aplicações é o ECOSystem [ZEN 02]. Este sistema atribui preços que as aplicações precisam pagar para utilizar cada recurso necessário (e.g., acesso a memória, rede ou disco). As aplicações pagam estes acessos através de uma “moeda”, chamada *currentcy*, que o sistema distribui periodicamente. Isto permite o desenvolvimento de aplicações que adaptam seu fluxo de execução com base em seu saldo de *currentcy*.

Uma série de trabalhos também foram realizados para estabelecer interfaces de programação (API) para gerência de energia pelas aplicações. Estes trabalhos focam em interfaces que fornecem à aplicação informações acerca do estado de componentes, bem como permitem à aplicação influenciar a gerência de energia do sistema operacional, geralmente através de "dicas" passadas ao sistemas operacional (*Application Hints*). Os projetos *PASA* [PER 02] e *Cooperative IO* [WEI 02b] permitem à aplicação passar ao sistema operacional suas restrições temporais, informando, por exemplo, prazos para chamadas de funções e expectativas de tempo de execução. Esta informação é então utilizada pelo gerente de energia para melhorar suas previsões sobre o comportamento futuro das aplicações, permitindo uma gerência de energia mais eficiente. Outro trabalho explora transformações no código da aplicação, realizadas através do compilador, para agrupar acessos a disco por uma tarefa, inserindo no código dicas informando o sistema operacional sobre quanto tempo estes acessos deveriam levar [HEA 04]. De posse desta informação, o sistema operacional pode implementar políticas que reduzem a velocidade de rotação do disco gerando um menor número de erros na previsão dos períodos de ociosidade.

Outro projeto que permite a passagem de dicas da aplicação para o sistema operacional é o *Ghosts in the machine* [ANA 04], porém a semântica das dicas passadas pela aplicação neste projeto são diferentes. Ao invés de passar informações sobre sua execução, aplicações adaptativas consultam informações sobre modo de operação atual e custos para trocas entre modos de operação dos dispositivos que utilizam. Baseada nesta informação, a aplicação decide realizar tarefas de um modo que consuma menos energia e informa o sistema operacional que foi necessário degradar o desempenho devido a um erro na previsão do comportamento da aplicação feita pela política de gerência de energia (e.g., um navegador Web necessita buscar um arquivo pela rede porque o mesmo arquivo não pode ser obtido de sua cache em disco, o que seria mais rápido, já que o disco havia sido desligado pela política de gerência de energia).

2.9 Interfaces de Gerência de Energia Dirigida Pela Aplicação em Sistemas Embarcados

Dois tipos de interfaces para aplicações foram estudadas neste capítulo. Primeiramente foram apresentadas interfaces que permitem ao programador da aplicação modificar o estado de operação de cada componente individualmente. Estas interfaces, implementadas por sistemas como o TINYOS [CUL 01], são simplistas (e.g., permitindo apenas ligar e desligar dispositivos). O segundo tipo de interface foram as desenvolvidas por trabalhos que exploram o envolvimento das aplicações na gerência de energia. APIs foram desenvolvidas, basicamente, para permitir às aplicações informar o gerente de energia sobre algumas de suas características (e.g., prazos, tempos de execução) ou para expôr à aplicação o estado atual dos componentes de hardware, permitindo o desenvolvimento de aplicações que se adaptem ao ambiente de execução.

Sistemas embarcados apresentam uma grande variedade de dispositivos que diferem tanto quanto às funcionalidades que implementam quanto aos recursos que oferecem para gerenciar seu consumo de energia. Para uma classe específica destes dispositivos, os chamados *sistemas profundamente embarcados* [2], estas diferenças se mostram ainda mais importantes, já que limitações de recursos (e.g., memória, capacidade de processamento), assim como restrições temporais de execução, muitas vezes inviabilizam a implementação de técnicas complexas para realizar a gerência de energia. Neste contexto, é interessante que sistemas operacionais forneçam interfaces que permitam à aplicação controlar, e não apenas influenciar, a gerência de energia.

Contudo, um problema encontrado ao entregar a responsabilidade sobre a gerência de energia do sistema à aplicação é o aumento da complexidade destas aplicações à medida que um número maior de componentes venham a ser utilizados. Este problema tende a piorar quando o sistema operacional aumenta a granularidade em que

[2] *Sistemas Profundamente Embarcados* são sistemas computacionais embarcados, de pequeno porte e com funcionalidade específica cujos requisitos de preço, tamanho e desempenho levam ao emprego de dispositivos de hardware extremamente simples, como microcontroladores que operam a baixas freqüências (muitas vezes na ordem de KHz) e que possuam pouca quantidade de memória (na ordem de KBytes).

seus componentes são organizados. Sistemas que implementam componentes desta maneira (e.g., EPOS [FRö 01]) normalmente o fazem de forma hierárquica, i.e., utilizam componentes para abstrair camadas de software. Estes sistemas se beneficiariam de um mecanismo que permita representar as interações necessárias entre componentes para realizar o processo de troca entre modos de operação, permitindo que a aplicação gerencie apenas os componentes de alto nível do sistema.

Capítulo 3

O sistema de Gerência de Energia Proposto

Neste capítulo é proposta uma infra-estrutura de software para permitir a gerência do consumo de energia dirigido pela aplicação em sistemas embarcados. A gerência de energia é realizada através de chamadas da aplicação a uma API (*Application Programming Interface*) uniforme que é implementada por todos os componentes do sistema operacional. Para garantir o correto funcionamento, as relações entre componentes do sistema foram especificadas através de Redes de Petri. Esta especificação permite não só uma análise em alto-nível dos procedimentos de troca de modo de operação dos componentes, mas também o estabelecimento de um mecanismo de troca de mensagens, em que os componentes se coordenam para garantir a consistência na mudança de estado dos serviços do sistema operacional (e.g., comunicação, processamento, sensoriamento) ou de todo o sistema.

Mecanismos convencionais para gerência do consumo de energia, tanto em sistemas operacionais de propósito geral (e.g, LINUX, WINDOWS) quanto em alguns sistemas operacionais embarcados (e.g., μCLINUX, VXWORKS), analisam dinamicamente o comportamento do sistema para determinar quando um dispositivo deve modificar seu modo de operação, passando a consumir menos energia. Outros sistemas para computação embarcada implementam apenas interfaces simples que muitas vezes não sa-

tisfazem as necessidades do programador da aplicação, que é obrigado a implementar na própria aplicação os mecanismos de gerência de energia (e.g., TINYOS, ECOS). A implementação do software responsável por gerenciar energia normalmente é feita através de iniciativas independentes de cada fabricante ou programador do sistema, que exportam APIs rígidas (*drivers*) e, em alguns casos, incompletas para os usuários.

Tentativas de padronização das interfaces de gerência de energia não foram largamente adotadas pela indústria de sistemas embarcados e, mesmo se houvesse sido, muitos problemas continuariam a existir, já que a padronização é feita num nível muito baixo de abstração. A maioria das plataformas de computação comerciais (computadores pessoais) implementam APM (*Advanced Power Management*) ou ACPI (*Advanced Configuration Power Interface*) para gerenciar a energia consumida pelo sistema. Estes mecanismos possibilitam colocar dispositivos ociosos em estados de baixa potência porém, como discutido na seção 2.7, isto é feito utilizando mecanismos custosos que também podem limitar a configuração dos dispositivos de hardware. Os problemas encontrados no emprego destas interfaces inviabilizam seu uso em sistemas embarcados devido a várias restrições, que vão desde a falta de recursos para suportá-las (e.g., processamento, memória) até requisitos funcionais, como disponibilidade ou restrições temporais.

Os estudos apresentados no capítulo 2 mostram várias tentativas recentes de dar às aplicações um papel mais ativo na gerência de energia. Tendo em vista que a maioria dos sistemas embarcados são dispositivos de finalidade específica que, geralmente, executam apenas uma aplicação, foi considerado que um bom lugar para determinar a estratégia de gerência do consumo de energia seria na própria aplicação. As seções a seguir detalham a proposta deste trabalho, apresentando a API definida para que a aplicação gerencie seu consumo de energia, o mecanismo de especificação das relações entre componentes do sistema e a estratégia de propagação das trocas de modos de operação.

3.1 Interface de Gerência do Consumo de Energia para Componentes de Software e Hardware

Na estratégia proposta neste trabalho é esperado que o programador da aplicação especifique, em seu código-fonte, quando certos componentes não estão sendo utilizados. Para isso, foi definida uma API uniforme de gerência do consumo de energia. A mesma interface permite interação da aplicação com o sistema (através de seus componentes), de componentes do sistema entre si, de componentes do sistema a dispositivos de hardware e, inclusive, o acesso direto das aplicações aos dispositivos de hardware. Para evitar que o programador tenha que, manualmente, acordar cada um dos componentes que utiliza, o mecanismo de gerência abstraído pela API garante que estes componentes retomem o seu estado anterior automaticamente quando utilizados.

A figura 3.1 apresenta todos estes modos de interação através de um diagrama de comunicação UML de uma instância hipotética de sistema. A aplicação pode acessar um componente global (`System`), que conhece todos os componentes instanciados no sistema, provocando a alteração do modo de operação de todo o sistema (seqüência de execução `1`). A API também pode ser acessada para alterar os modos de operação de grupos de componentes responsáveis por implementar funcionalidades específicas do sistema (no caso, comunicação através da seqüência de execução `2`). A aplicação ainda pode acessar diretamente o hardware, utilizando a API disponível nos *drivers*, como `NIC` (*Network Interface Card*), `CPU` e `ADC` (no caso, acesso ao componente `CPU` pela seqüência `3`). Como dito anteriormente, a API também é utilizada entre os componentes do sistema, como pode ser observado na figura 3.1 nas trocas de mensagens entre os componentes representados.

De modo a aliar portabilidade da aplicação à facilidade de desenvolvelas, foi decisão deste projeto dotar a interface de um conjunto mínimo de métodos e de uma estrutura que permita a configuração das características e dos modos de operação dos componentes, permitindo ao programador da aplicação configurar o gerente de energia conforme as necessidades de sua aplicação. Neste caso, a portabilidade vem do fato de

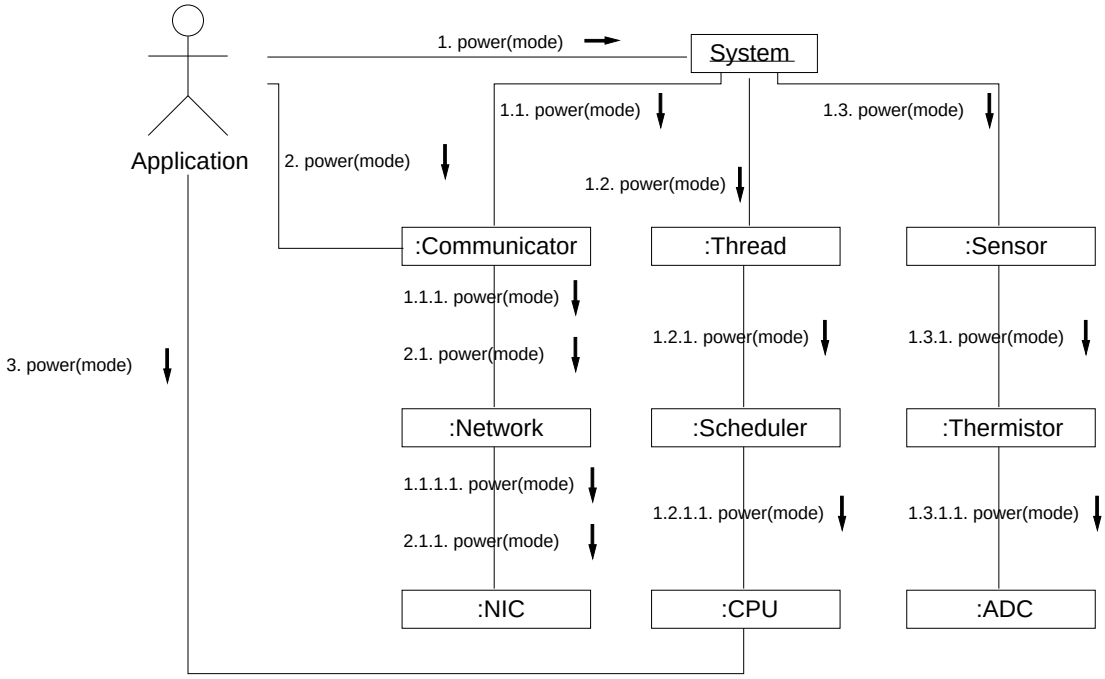


Figura 3.1: Meios de acesso à API de gerência do consumo de energia.

a aplicação não necessitar implementar procedimentos específicos para cada dispositivo de hardware ao alterar seus modos de operação. Estes procedimentos são abstraídos pela API. Já a facilidade de desenvolvimento ocorre porque o programador da aplicação não necessita analisar os manuais do hardware a fim de identificar os modos de operação disponíveis, os procedimentos para realizar as trocas e as conseqüências de cada uma destas mudanças.

Focando a simplicidade, apenas dois métodos foram definidos para a API: um para alterar o modo de operação e outro para consulta-lo. Além dos métodos, a API ainda contém uma relação dos modos de operação disponíveis em cada componente. Esta relação não possui um tamanho fixo, já que cada componente deve enumerar nela todos os modos de operação possíveis. Componentes de hardware de baixa potência que são utilizados em sistemas embarcados freqüentemente apresentam um grande conjunto de modos de operação. Aplicações embarcadas se beneficiam disso para utilizar os modos de operação que melhor se adaptam em determinadas situações. A mesma aplicação, contudo, muito dificilmente utilizará todos os modos de operação disponíveis. Para per-

mitir que o programador da aplicação selecione os modos que sua aplicação utilizará, sem precisar conhecer profundamente o hardware sendo utilizado, foram definidos modos de operação universais. Estes modos universais, chamados de `FULL`, `LIGHT`, `STANDBY` e `OFF`, estão originalmente amarrados a modos de operação equivalentes aos modos mais comumente utilizados em cada componente. Esta amarração tenta seguir a mesma semântica para todos os componentes sempre que possível (i.e., sempre que modo equivalente existir para o componente em questão). O programador da aplicação pode, contudo, configurar estas amarrações para utilizar os modos de operação que desejar.

A Tabela 3.1 apresenta a semântica pretendida para os modos de operação. Quando o dispositivo está operando com toda sua capacidade, ele está no modo `FULL`. Neste modo de operação, o sistema configura o dispositivo para operar fornecendo seus serviços da maneira mais completa possível, incluindo todas as suas funcionalidades, porém consumindo mais energia. O modo `LIGHT` coloca o dispositivo em um modo de operação onde ele continua oferecendo todas as suas funcionalidades, porém consumindo menos energia e, muito provavelmente, implicando em perda de desempenho. Alguns dispositivos, contudo, não apresentam tais modos. Nestes casos, o modo `LIGHT` estará associado ao modo de operação com maior número de funcionalidades disponível ou a um modo de menor consumo de energia que seja comumente utilizado pelo componente em questão. A fim de evitar erros por parte do programador da aplicação, estas situações estarão claramente especificadas na documentação do gerente de energia. Exemplos destes modos estão presentes em dispositivos que possuem seleção de diferentes níveis de tensão e/ou divisores de freqüência de operação (DVS - *Dynamic Voltage Scaling*). O retorno deste modo de operação para o modo `FULL` é rápido e, geralmente, não implica em atrasos consideráveis para a aplicação.

Para o caso específico de DVS em processadores, o modo `FULL` é associado ao modo de operação de mais alta freqüência, e o modo `LIGHT` é associado ao modo de operação de mais baixa freqüência. Entre os modos `FULL` e `LIGHT` existem modos de operação que realizam saltos de 10% na freqüência de operação do dispositivo. Estes modos são `DVS_90`, `DVS_80`, `DVS_70`, `DVS_60`, `DVS_50`, `DVS_40`, `DVS_30`, `DVS_20`. Os modos `DVS_100` e `DVS_10` são equivalentes aos modos `FULL` e `LIGHT`, respecti-

vamente. Caso o processador em questão não permita a configuração da freqüência de operação a uma granularidade tão fina, ou as freqüências disponíveis não implementem saltos de 10%, os modos DVS devem sempre corresponder à freqüencia disponível que seja imediatamente maior.

Modo	**FULL**	**LIGHT**	**STANDBY**	**OFF**
Energia	Alto	Baixo	Baixíssimo	Nenhum
Funcionalidades	Total	Limitada	Nenhuma	Nenhuma
Desempenho	Máximo	Reduzido	Parado	Parado

Tabela 3.1: Semântica dos modos de operação universais.

Nos modos STANDBY e OFF o dispositivo para de operar. Entretanto, quando em STANDBY, o dispositivo está em um modo do qual pode voltar a operar normalmente quando necessário, podendo continuar sua operação do ponto em que parou. Embora parado, neste modo o hardware ainda consome uma pequena quantidade de energia. Esta energia é necessária para manter dados em memória e registradores, permitindo que o dispositivo volte a operar sem que ocorra uma reinicialização. Já no modo OFF o dispositivo é desligado. Quando isto é feito, o dispositivo perde sua configuração original, e seu retorno a um modo operacional implica em uma reinicialização. Outra diferença importante entre os dois modos é o tempo para que o componente retorne a um modo ativo. Como o retorno do modo OFF implica em uma reinicialização, este processo tende a ser muito mais lento que o retorno do modo STANDBY.

Além dos requisitos funcionais, também é desejável que a API seja de fácil manutenção e aplicável a sistemas já existentes. Sendo a gerência do consumo de energia uma propriedade não-funcional no âmbito de sistemas operacionais [LOH 05], foi considerado importante implementar esta API utilizando programação orientada a aspectos [KIC 97], permitindo, assim, o isolamento do gerente de energia do restante do sistema.

A técnica de programação orientada a aspectos define estruturas que

permitem aos programadores separar características não funcionais cujas implementações implicam em replicação de código em várias partes do sistema. Linguagens de programação que suportam aspectos definem estruturas para especificações de "comportamentos adicionais"(*advice*) e "pontos de execução"(*join point*), que definem, geralmente através de expressões regulares, os pontos do código alvo (i.e., o sistema existente) onde os comportamentos adicionais devem ser inseridos.

Os pontos de execução de um aspecto devem especificar a posição em que os comportamentos adicionais serão incluídos, isto é, se o comportamento adicional é inserido antes (*before*), depois (*after*) ou ao redor (*around*) do ponto definido. Um exemplo de utilização de aspecto para implementar uma característica não-funcional de sistemas operacionais é sincronização. Um aspecto definiria pontos de execução que representam as regiões críticas do código, e um comportamento adicional atuaria "ao redor"destes pontos de execução realizando operações `lock` num mutex ao entrar nas regiões críticas e `unlock` ao sair.

O problema encontrado ao abstrair a gerência de energia como um aspecto é a questão da independência arquitetural que se espera encontrar em um aspecto de software. Mais especificamente, é interessante que um aspecto possa ser aplicado a qualquer componente do sistema. Num aspecto gerente de energia o controle das trocas de modo de operação poderia ser generalizada para ser aplicada a qualquer componente. Porém, não existe um modo genérico de implementar as ações que realmente trocam os modos de operação de um componente, tanto de software, quanto de hardware. Isto ocorre porque esta implementação tem que levar em conta detalhes arquiteturais e/ou de implementação específicos de cada sistema ou dispositivo. Para permitir que um aspecto de gerência de energia seja aplicado aos componentes de um sistema operacional, é estabelecido nesta proposta um mecanismo que representa as ações necessárias para trocar os modos de operação dos componentes. Este mecanismo, descrito na próxima seção, utiliza Redes de Petri Hierárquicas para especificar as interações que o gerente de energia precisa realizar com o sistema a fim de realizar as trocas de modo de operação de um determinado componente.

3.2 Redes de Troca de Modos de Operação

Nesta seção é descrita a parte da proposta que introduz o mecanismo de especificação das trocas de modo de operação. A ferramenta escolhida para modelar estas transições foi Redes de Petri. Redes de Petri se mostraram uma boa escolha devido à clareza de sua representação gráfica e ao extenso leque de modelos de análise matemática existentes. Outra característica das Redes de Petri explorada nesta proposta é a representação hierárquica que esta ferramenta possui, o que permite abstrair interações entre componentes [PET 77]. Como ficará claro ao longo desta seção, as análises matemáticas permitiram provar a inexistência de *deadlocks*, além de demonstrar a alcançabilidade dos estados desejados (através das marcações pretendidas) e a impossibilidade de estados indesejados serem alcançados, e a hierarquia de Redes de Petri facilitou a organização e representação destas redes.

Embora os procedimentos para realizar as trocas de modo de operação dos componentes sejam diferentes (tanto para software, quanto para hardware), o controle destas trocas pode ser expresso de forma generalizada. Para isso, foi especificada uma rede de troca de modos de operação que determina condições para que as trocas ocorram. Neste trabalho, a proposta da rede de troca de modos de operação generalizada é baseada nos modos universais definidos na seção anterior (`FULL`, `LIGHT`, `STANDBY` e `OFF`). Esta rede pode, contudo, ser extendida para atender uma quantidade maior de modos de operação quando necessário.

Devido ao tamanho da rede definida, uma versão simplificada foi produzida para apresentar a seqüência de transições realizadas supondo que o componente sendo manipulado estivesse saindo do modo de operação `OFF` para `FULL` (figura 3.2). A rede de Petri completa, ou seja, incluindo todos os modos de operação, pode ser vista no anexo A. Como pode ser observado na figura, há lugares associados aos modos de operação existentes (`FULL` e `OFF`). Um recurso nestes lugares marca o modo de operação atual do componente.

O lugar `Atomic_Execution` é responsável por garantir que operações diferentes de troca de modo de operação não sejam executadas em paralelo. Para

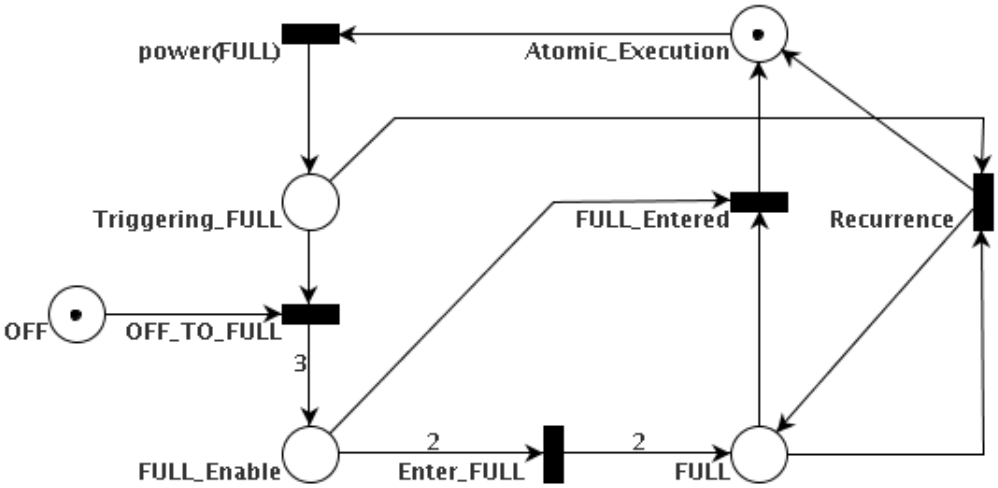


Figura 3.2: Comportamento da rede generalizada de troca de modos de operação.

isto, este lugar é sempre inicializado com um recurso. Este recurso é necessário para habilitar as transições que disparam as mudanças de modos de operação. A partir do momento que uma chamada é realizada à API de gerência de energia solicitando a troca de modo de operação, uma das transições que representam estas chamadas à API é disparada (no exemplo, `power(FULL)`), consumindo o recurso de `Atomic_Execution` e impedindo que as transições que dão início às demais trocas de modo de operação sejam disparadas (verificar rede completa no anexo A). Além disso, um novo recurso inserido no lugar `Triggering_FULL` permite a habilitação das transições que removem o recurso que marca o modo de operação atual do componente (no lugar `OFF`). Como o componente do exemplo está no modo `OFF`, apenas a transição `OFF_TO_FULL` é habilitada. A partir do seu disparo, o recurso que marcava o lugar `OFF` é consumido, e três recursos são inseridos no lugar `FULL_Enable`. Isto é feito para habilitar a transição `Enter_FULL`, que é responsável por executar as operações necessárias para que o componente seja efetivamente colocado no modo de operação desejado. Após o disparo desta transição, dois recursos são inseridos no lugar `FULL`, habilitando a transição `FULL_Entered`, que finaliza o processo consumindo o recurso restante do lugar `FULL_Enable` e retornando um recurso para o lugar `Atomic_Execution`. Ao final do procedimento, um recurso foi retirado do lugar `OFF` e inserido no lugar `FULL`. Para evitar a ocorrência de *deadlocks* ao

disparar transições que levem ao modo de operação atual do componente, uma outra transição teve que ser inserida (`Recurrence`). Esta transição retorna o recurso tomado do lugar `Atomic_Execution` em caso de recorrência (e.g., solicitar disparo da transição `power(FULL)` quando já houver um recurso no lugar `FULL`).

A rede de troca de modos de operação generalizada foi submetida a uma ferramenta de análise [AKH 05], o que permitiu verificar algumas propriedades da rede implementada:

Vivacidade: A análise da vivacidade desta rede permite determinar a existência ou não de *deadlocks*. Para isso, é necessário que todas as transições da rede em questão sejam quase-vivas. Uma transição é dita quase-viva quando existe uma seqüência de disparo de transições a partir de uma marcação inicial que conduzirá a seu disparo. Neste caso, esta rede de Petri pode ser considerada livre de *deadlock* a partir da marcação inicial utilizada, ou seja, um recurso depositado no lugar `Atomic_Execution` e outro em um dos lugares referentes aos modos de operação (`FULL`, `LIGHT`, `STANDBY` ou `OFF`).

Alcançabilidade: A análise de alcançabilidade permite identificar as marcações possíveis de serem atingidas a partir de uma marcação inicial. Esta análise é feita através da montagem de um grafo de alcançabilidade, que inclui todos os possíveis estados da rede. Para esta rede, a análise de alcançabilidade pôde ser completa, já que seu grafo de alcançabilidade resultou em um número finito de estados. Assim, foi possível concluir que todos os estados desejados foram alcançados, e que estados indesejados (e.g., recursos presentes em mais de um lugar referente a modo de operação simultaneamente) não são possíveis de serem alcançados.

A rede generalizada representa a troca de modos de operação sob uma visão de alto nível, onde as especificidades envolvidas nestas trocas não estão expressas. Logo, para tornar possível a inferência dos procedimentos de troca de modos de operação a partir desta rede, um refinamento é necessário. Este refinamento é feito explorando a característica hierárquica das Redes de Petri. Através desta característica, uma rede

inteira pode ser substituída por um lugar ou transição para modelar um nível mais abstrato (abstração), ou lugares e transições podem ser substituídos por sub-redes para prover um modelo mais detalhado (refinamento).

A figura 3.3 apresenta um exemplo de refinamento para aplicação da rede generalizada no controle dos modos de operação do componente `CMAC`. Para refinar os procedimentos de troca de modo de operação na rede generalizada, as transições "`Enter`" são substituídas por subredes que implementam os procedimentos de troca de modo de operação com maior detalhe. No exemplo, é apresentada a subrede que implementa a mudança do componente `CMAC` para o modo de operação `FULL`. Para formar a rede de troca de modos de operação deste componente, esta subrede substitui a transição `Enter_FULL` na rede generalizada. Esta subrede ainda apresenta transições que abstraem o disparo das trocas de modo de operação de outros componentes.

O `CMAC` é uma implementação em software de um MAC (*Media Access Control*) configurável para um módulo de rede de sensores sem fio [HIL 00]. Neste dispositivo, a comunicação entre o processador e o rádio é realizada através de um barramento serial (SPI). Neste exemplo, é esperado que a aplicação utilize a API do componente `CMAC` como interface das funcionalidades de comunicação. Quando a aplicação realiza uma chamada à função `CMAC::power(FULL)`, por exemplo, o procedimento para ligar as funcionalidades de comunicação deve iniciar ligando o dispositivo de rádio (`Radio::power(FULL)`), passado ao ligamento do dispositivo responsável pela comunicação com o tranceptor de rádio (`SPI::power(FULL)`). A partir deste ponto o mecanismo de envio de dados já está inicializado. Já o mecanismo de recepção necessita de um temporizador, que é utilizado para estabelecer o *beacon* de verificação da existência de portadoras na freqüência de operação do rádio. Após a inicialização do temporizador (`Timer::power(FULL)`), o `CMAC` está ativo. A próxima seção descreve como as redes de troca de modo de operação são montadas para permitir a propagação destas trocas entre os componentes do sistema.

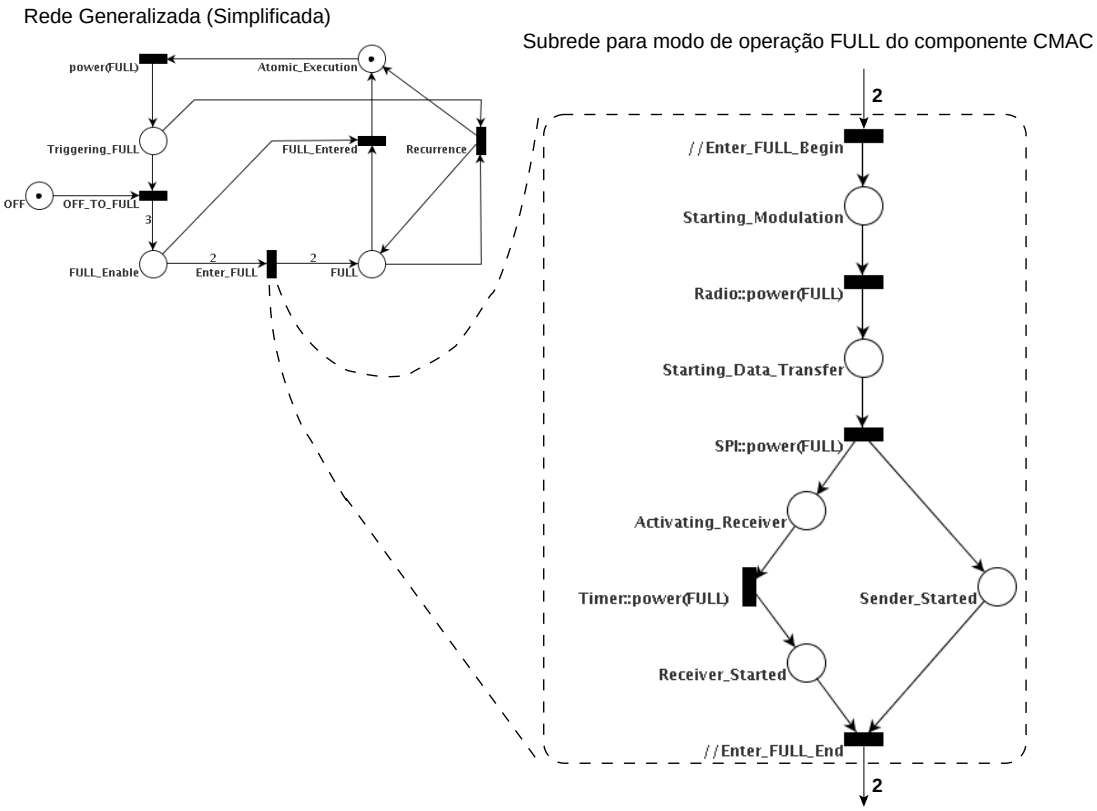


Figura 3.3: Refinamento com rede de Petri hierárquica para o componente CMAC.

3.3 Propagação de Mensagens

Conforme as aplicações embarcadas crescem em complexidade, elas passam a usar um maior número de componentes de sistema. Com isso, pode se tornar impraticável para programadores de aplicação controlar o consumo de energia de cada componente individualmente. Na maioria das vezes esta complexidade está evidente na aplicação quando ela faz uso de vários componentes diretamente. Contudo, mesmo aplicações extremamente simples podem utilizar um conjunto complexo de componentes.

Como exemplo, a figura 3.4 apresenta uma aplicação hipotética em que a gerência do consumo de energia é feita pela aplicação. A aplicação utilizada implementa um módulo de telemetria que envia a leitura de um sensor de pressão a cada dois segundos através de um modem GPRS. Na figura 3.4(a) fica claro a complexidade existente ao se oferecer somente APIs para componentes isolados do sistema. Neste exemplo, antes de desligar o modem, a pilha de comunicação TCP/IP precisa ser "desligada", ou seja, precisa ter todos os dados enviados antes que o modem possa ser desligado. Após o desligamento do modem, ainda é desejável que uma das portas seriais (UART) também seja desligada, já que é através desta porta que o processador realiza a comunicação com o modem. Neste caso, como o exemplo é hipotético, não foram considerados outros componentes que poderiam estar sendo utilizados (e.g., um temporizador para controlar *time-outs* no protocolo de comunicação). Complexidades semelhantes a esta estão presentes em quase todos componentes de alto-nível de um sistema. Abstrair estes detalhes do programador de uma aplicação embarcada melhora em muito a usabilidade da API, como mostram a s figuras 3.4(b) e 3.4(c).

O restante desta seção é dedicada a apresentar como esta proposta explora a organização hierárquica das redes de troca de modos de operação para identificar as relações entre componentes e propagar mensagens para coordenar as trocas de modo de operação.

```
void event() {
 static Modem modem;
 static Pressure_Sensor sensor;
 int pressure = sensor.sample();
 modem.send(dest, &pressure, 2);
 //Stoping communication
 Network::flush();
 modem.power(STANDBY);
 UART::power(STANDBY);
 //Stoping sensor
 sensor.power(STANDBY);
 ADC::power(STANDBY);
}

int main() {
 Alarm alarm(2000000, &event);
 while(1) {
  //Stoping processing
  CPU::power(STANDBY);
 }
}
```

(a) Controlando todos componentes

```
void event() {
 static Modem modem;
 static Pressure_Sensor sensor;
 int pressure = sensor.sample();
 modem.send(dest, &pressure, 2);
 //Stoping communication

 modem.power(STANDBY);

 //Stoping sensor
 sensor.power(STANDBY);

}

int main() {
 Alarm alarm(2000000, &event);
 while(1) {
  //Stoping processing
  CPU::power(STANDBY);
 }
}
```

(b) Controlando subsistemas

```
void event() {
 static Modem modem;
 static Pressure_Sensor sensor;
 int pressure = sensor.sample();
 modem.send(dest, &pressure, 2);
}

int main() {
 Alarm alarm(2000000, &event);
 while(1) {
  //Stoping everything
  System::power(STANDBY);
 }
}
```

(c) Controlando todo o sistema

Figura 3.4: Aplicações hipotéticas com gerência do consumo de energia dirigido pela aplicação.

3.3.1 Propagação Hierárquica de Mensagens

Para que um conjunto de componentes envolvido na implementação de alguma funcionalidade seja desativado ou colocado em modos de operação mais restritos de forma eficiente é necessário garantir que os artefatos de software e hardware primeiramente finalizem as atividades iniciadas, ou que se adaptem aos novos parâmetros de operação. Da mesma forma, é necessário garantir que estes componentes possam operar corretamente ao retornar para modos de operação funcionais. Para tanto, é necessário estabelecer um mecanismo pelo qual os componentes possam interagir, além de um meio pelo qual estas interações possam ser especificadas.

Dadas as definições já apresentadas (API e redes de troca de modos de operação), é possível definir os procedimentos de troca de modo de operação para cada componente. Neste contexto, o mecanismo de interação é formado por trocas de mensagens através da API. Já a seqüência em que ações devem ser executadas ou mensagens devem ser propagadas pode ser derivada das Redes de Petri de cada componente. Analisando a rede da figura 3.3, é possível notar a existência de transições que disparam trocas de modos de operação em outros componentes (`Radio::power(FULL)`, `SPI::power(FULL)` e `Timer::power(FULL)`). Estas transições constituem os pontos em que há trocas de mensagens entre componentes. Utilizando a propriedade hierárquica das Redes de Petri para realizar todas as substituições existentes, a rede de troca para o modo `FULL` do componente `CMAC` seria a rede apresentada na figura 3.5.

A partir da análise das redes é possível montar, em tempo de compilação, os métodos que garantirão a seqüência correta de execução dos procedimentos de troca de modos de operação. No exemplo do `CMAC` (figura 3.5) as três conexões com outras redes (`Timer`, `SPI` e `Radio`) indicam o ponto onde há troca de mensagens entre os componentes, ou seja, onde há a propagação das trocas de modo de operação. Realizando a análise desta rede é possível extrair um procedimento algoritmo como o descrito pelos diagramas de atividades apresentados na figura 3.6. A atividade `Power` representa o comportamento esperado para controle das trocas de modo de operação pela rede de troca de modo de operação generalizada (anexo A). As outras atividades representam o

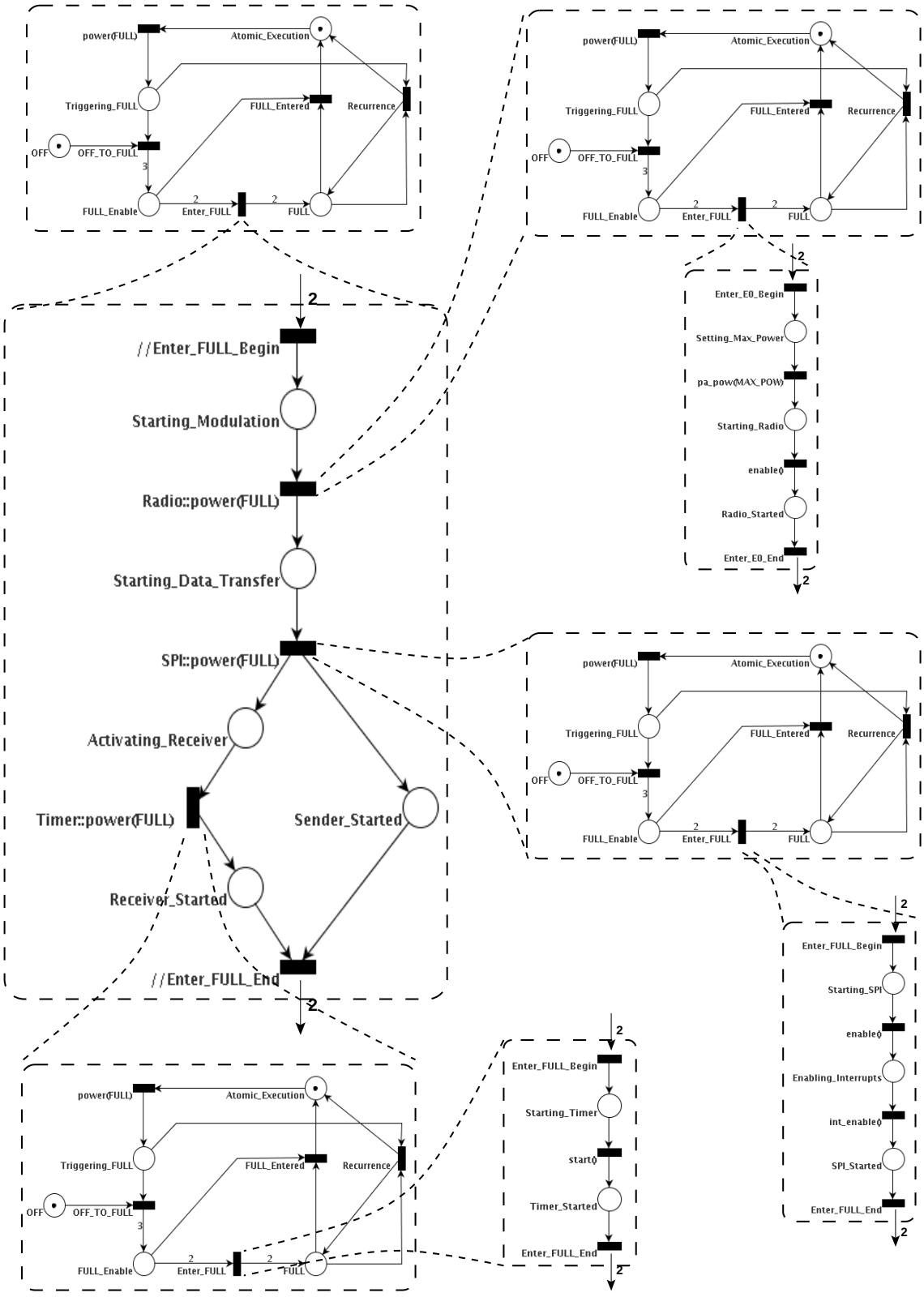


Figura 3.5: Rede completa de troca para o modo FULL do componente CMAC.

comportamento esperado do código extraído das redes dos componentes envolvidos na comunicação do sistema. Por exemplo, na atividade `CMAC Entra FULL` há chamadas para a atividade `Power` dos componentes que o `CMAC` utiliza em sua implementação, representando os pontos onde há propagação das trocas de modo de operação.

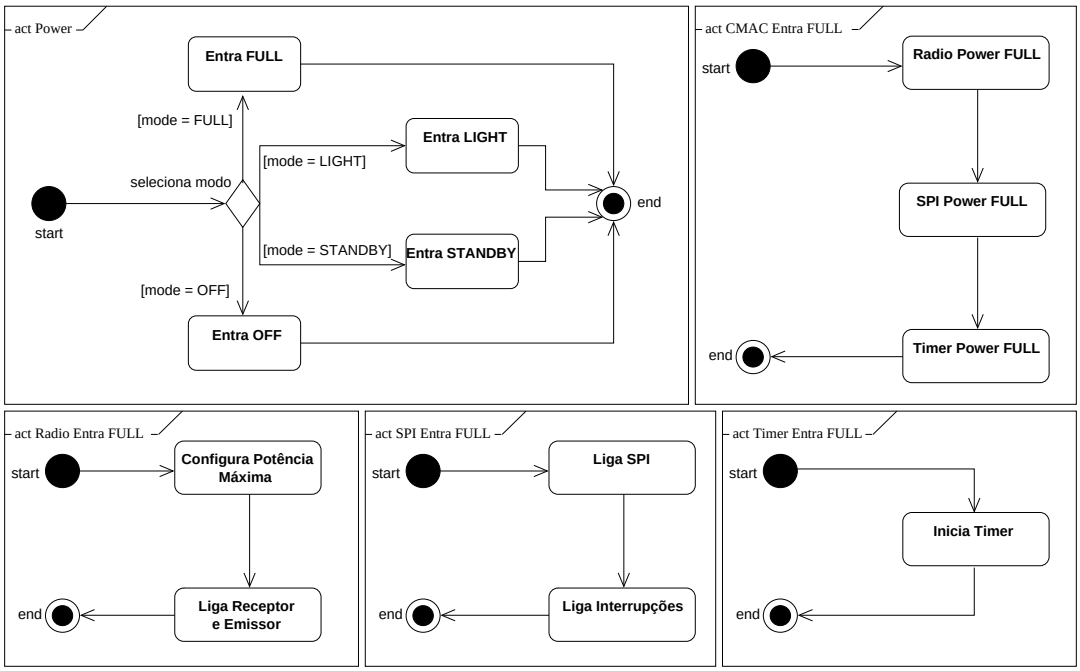


Figura 3.6: Procedimentos para troca de modo de operação.

Propagação para todo o sistema

Ações de gerência do consumo de energia do sistema como um todo são tratadas por um componente global do sistema (`System`). Este componente contém referências para todos os componentes em uso pela aplicação. Então, se uma aplicação deseja alterar o modo de operação do sistema inteiro, isto pode ser feito acessando a API deste componente, que propagará este pedido para os demais componentes. Isto será feito através de uma lista montada em tempo de execução através do aspecto de gerência de energia, que utilizará as chamadas de construção e destruição de componentes para, respectivamente, incluir e remover referências a instâncias de componentes desta lista. Quando a API de gerência do consumo de energia do sistema é acessada pela aplicação,

o sistema realiza uma varredura pela lista de instâncias que possui, disparando chamadas às APIs dos componentes que registrou.

3.3.2 Compartilhamento de Recursos

O compartilhamento de recursos é uma característica de sistemas computacionais que precisa ser tratada nesta proposta. Problemas podem ocorrer na troca de modos de operação quando componentes de alto nível compartilham o mesmo componente de hardware. Por exemplo, uma aplicação que utiliza dois sensores que compartilham o mesmo conversor analógico-digital (ADC) não pode ter o ADC desligado devido à solicitação de um dos sensores se o outro sensor ainda o está utilizando (figura 3.7).

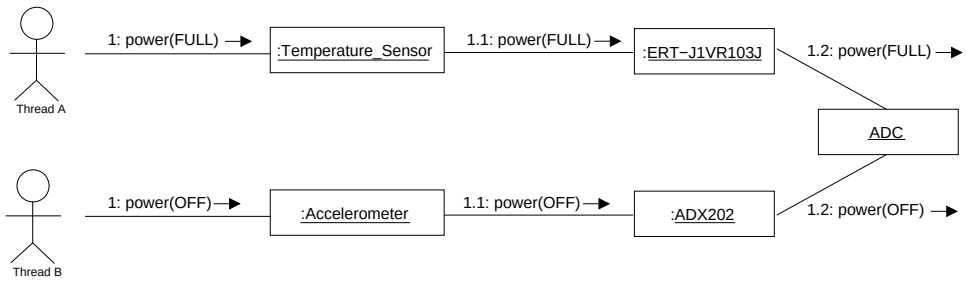


Figura 3.7: Dois sensores diferentes compartilham o mesmo ADC.

Para resolver este problema, foi adotado um mecanismo de *contadores de uso*. Cada componente compartilhado possui um contador de referências para cada modo de operação. Através deste mecanismo é possível saber quantos “usuários” (i.e., outros componentes do sistema ou aplicações) utilizam cada componente. Como os modos de operação de cada componente estão organizados para representar estados de maior consumo/desempenho para menor consumo/desempenho (FULL–OFF), os componentes do sistema permanecem no modo de operação com maior número de funcionalidades que possuir “usuários”. Para isto, sempre que uma requisição de troca de modos de operação é feita, os contadores do componente são atualizados (contador do modo atual decrementado e do modo pretendido incrementado) e um teste é realizado verificando o somatório dos contadores para os modos de operação “maiores” que o pretendido, ou seja, mo-

dos de operação menos restritos. Se o somatório for zero, o componente entra no modo pretendido. Se houver usuários para modos de operação menos restritos, o componente permanece como está até que novas solicitações de troca de modo de operação sejam realizadas.

As propostas apresentadas neste capítulo definem as características desejáveis de um gerente de energia dirigido pela aplicação para sistemas embarcados. Uma API configurável de gerência de energia foi definida e um mecanismo baseado em Redes de Petri foi utilizado para especificar os procedimentos para troca de modo de operação dos componentes do sistema. O próximo capítulo apresenta como esta proposta foi implementada no EPOS, um sistema operacional baseado em componentes para plataformas embarcadas.

Capítulo 4

Implementação do Gerente de Energia Proposto

Este capítulo apresenta e discute a implementação de um protótipo do gerente de energia proposto neste trabalho. Primeiramente é apresentado o ambiente experimental de software e hardware utilizado. A seguir, é apresentada a solução desenvolvida para a implementação do gerente de energia como um aspecto e para a geração automática de código a partir das redes de modos de operação.

4.1 Ambiente Experimental

O gerente de energia foi implementado no sistema operacional EMBEDDED PARALLEL OPERATING SYSTEM (EPOS) [FRö 01]. O EPOS é um framework baseado em componentes que permite a geração de sistemas operacionais adaptados para aplicações dedicadas. Implementações deste sistema para dois microcontroladores foram utilizados (ATMEGA16 e ATMEGA128). Além disto, suporte para outros componentes presentes em algumas das plataformas utilizadas foi desenvolvido em conjunto com um trabalho paralelo que explorou suporte de sistema operacional para aplicações de redes de sensores sem fio [WAN 06]. Esta seção apresenta este ambiente experimental em maior detalhe.

4.1.1 Embedded Parallel Operating System - EPOS

O sistema operacional EPOS foi proposto por Fröhlich como protótipo para provar os conceitos de sua metodologia de projeto de sistemas orientados a aplicação (*Application-Oriented System Design* - AOSD) [FRö 01]. Esta metodologia utiliza várias técnicas avançadas de engenharia de software e programação que, combinadas, permitem a geração de sistemas especializados para aplicações dedicadas. Desde sua criação, o EPOS tem servido como ambiente para validação e extensão dos conceitos desta metodologia.

O EPOS oferece aos programadores de aplicação um framework baseado em componentes que permite a configuração e a geração de sistemas operacionais adaptados para aplicações dedicadas, isto é, sistemas específicos contendo apenas os componentes necessários para suportar a aplicação em questão. O framework do EPOS oferece ao programador da aplicação um conjunto de componentes de alto-nível, chamados de *Abstrações*, que implementam funcionalidades abstratas como `Thread`, `Communicator` e `Sensor` de forma independente de arquitetura. Estes componentes utilizam outros componentes deste sistema, chamados de *Mediadores*, que abstraem os dispositivos de hardware e exportam as funcionalidades destes dispositivos através de uma interface uniforme, possibilitando a independência arquitetural das *Abstrações*. Além dos componentes do sistema o EPOS também utiliza *aspectos* [KIC 97] e *características configuráveis* [CZA 98]. O primeiro permite a fatoração de características não-funcionais do sistema (e.g., compartilhamento, identificação) em artefatos de software independentes, o segundo permite que os componentes do sistema sejam configurados. O uso de aspectos e de características configuráveis aliados a técnicas de programação avançadas como *meta-programação estática* e *programação orientada a aspectos* disponibiliza à aplicação um sistema altamente configurável e adaptativo.

O sistema EPOS foi utilizado não somente por ter sido desenvolvido dentro do mesmo grupo de pesquisa deste trabalho, mas também por possuir características interessantes para a implementação da proposta. Dentre estas características cabe citar:

- A organização do sistema em componentes de software, que permite o desenvolvimento das redes hierárquicas de troca de modos de operação de forma modular, diminuindo sua complexidade.

- O uso de interfaces bem definidas para componentes do sistema, que facilitam a interação com os componentes do sistema, também facilitando o desenvolvimento das redes de troca de modos de operação.

- O fato deste sistema permitir a implementação de componentes de software como *aspectos*. No EPOS aspectos são implementados através da técnica chamada *Adaptadores de Cenário* [D'A 05], que implementa os aspectos utilizando técnicas de meta-programação estática e sem o uso de *weavers* de código.

4.1.2 Ambiente de Hardware

Durante o desenvolvimento deste trabalho o sistema EPOS foi portado para a arquitetura de processadores AVR em conjunto com outro projeto, que explorou suporte de sistema operacional para redes de sensores sem fio. Sendo assim, o protótipo de gerente de energia desenvolvido foi implementado e testado em processadores desta arquitetura e utilizando módulos de sensoriamento. Esta seção descreverá o ambiente de hardware utilizado.

Microcontroladores AVR

Os microcontroladores AVR são produzidos pela ATMEL [ATM 04a]. Estes microcontroladores RISC (*Reduced Instruction Set Computer*) de 8 bits são baseados na arquitetura de Harvard, ou seja, possuem barramentos separados para instruções e dados. Há vários modelos destes microcontroladores, cada um oferecendo uma combinação diferente de periféricos como temporizador, UART (*Universal Asynchronous serial Receiver and Transmitter*), SPI (*Serial Peripheral Interface*) e ADC (*Analog to Digital Converter*).

Para os protótipos deste trabalho foram utilizados microcontroladores

AVR dos modelos ATMEGA128L e ATMEGA16. No que tange o controle do consumo de energia, estes microcontroladores apresentam seis diferentes modos de operação de baixo consumo de energia (*sleep modes*). Como pode ser observado na tabela 4.1.2, a escolha do modo de operação correto para utilizar pode impactar bastante no consumo de energia, já que a potência nos diferentes modos de operação apresenta uma grande variação.

Modo de operação	Potência	Corrente
`idle`	14.28 mW	4.8 mA
`power save`	0.79 μW	266 μA
`power down`	0.71 μW	239 μA

Tabela 4.1: Potência e corrente drenada para alguns dos modos de operação do ATMEGA128. Adaptada de Kellner [KEL 06].

Módulos de Sensoriamento: Mica Mote2

A plataforma MICA2 é um módulo de sensoriamento para redes de sensores sem fio desenvolvido pela CrossBow [CRO 06]. Basicamente, esta plataforma é um dispositivo alimentado por baterias composto por um microcontrolador Atmel ATMEGA128, um *transceiver* de rádio Chipcon CC1000 [AS 04] e um conjunto de sensores (temperatura, luminosidade, aceleração, etc). Suporte de sistema operacional através do EPOS para este módulo de sensoriamento foi desenvolvido por Wanner [WAN 06].

O dispositivo de comunicação deste módulo, o CC1000, é um *transceiver* de radio freqüência de baixíssimo consumo de energia. Este dispositivo apresenta uma grande variedade de características que podem ser ajustadas para atender necessidades específicas de cada aplicação. Estas mesmas características permitem também configurar o dispositivo para consumir ainda menos energia. Dentre o que pode ser configurado neste *transceiver* estão a freqüência de operação (de 300 a 1000 MHz), potência de envio (de -20 a 10 dBm) e tensão de alimentação (de 2.1 a 3.6 V). Além destas características, o

CC1000 ainda apresenta um modo de espera (*Power Down Mode*) e recursos para ligar e desligar os módulos de envio (TX) e recepção (RX) separadamente. Diferentes modos de operação para este dispositivo apresentam diferenças consideráveis no consumo de energia, como pode ser observado na tabela 4.1.2, que relaciona potência e corrente drenada para alguns dos modos de operação do *transceiver*.

Modo de operação	Potência	Corrente
Power Down	0.66 μW	0.2 μA
Receive	31.7 mW	9.6 mA
Receive with polling	0.32 mW	96 μA
Transmit (-20 dBm)	28.4 mW	8.6 mA
Transmit (-5 dBm)	45.5 mW	13.8 mA
Transmit (0 dBm)	54.5 mW	16.5 mA
Transmit (5 dBm)	83.8 mW	25.4 mA

Tabela 4.2: Potência e corrente drenada do CC1000 operando a uma tensão de 3.3 V e frequência de transmissão de 868 MHz.

O MICA2 também possui um grande conjunto de sensores disponíveis. Estes sensores apresentam características variadas e, geralmente, consomem uma quantidade razoável de energia. Por exemplo, o sensor de temperatura presente nesta plataforma, que é um termistor, ou seja, um resistor cuja resistência varia conforme a temperatura, pode apresentar uma potência equivalente à apresentada pelo rádio quando enviando dados em seu modo de operação mais potente. O dispositivo presente na plataforma em questão é um `ERT-J1VR103J` da Panasonic [PAN 04]. A potência deste dispositivo, dependendo da temperatura e, conseqüentemente, da resistência assumida, pode atingir até 100 mW, o que é preocupantemente alto. Para evitar consumo de energia excessivo, o MICA2 disponibiliza pinos de controle que permitem inibir a corrente que alimenta estes sensores, permitindo "ligá-los" apenas quando necessário.

4.2 Extensão dos Componentes Para Inclusão da API

Como dito anteriormente, no sistema EPOS aspectos são aplicados aos componentes através da técnica chamada *adaptadores de cenário*. A figura 4.1 apresenta a estrutura de um adaptador de cenário no EPOS. O aspecto (cenário) implementa métodos que realizam as ações que devem ser incluídas antes ou depois do método original através dos métodos `enter` e `leave`. Por este mecanismo também é possível estender a interface do componente alvo. Para isto basta apenas incluir um novo método ao cenário e este método é adicionado à interface do componente através da herança existente entre `Scenario` e `Adapter`.

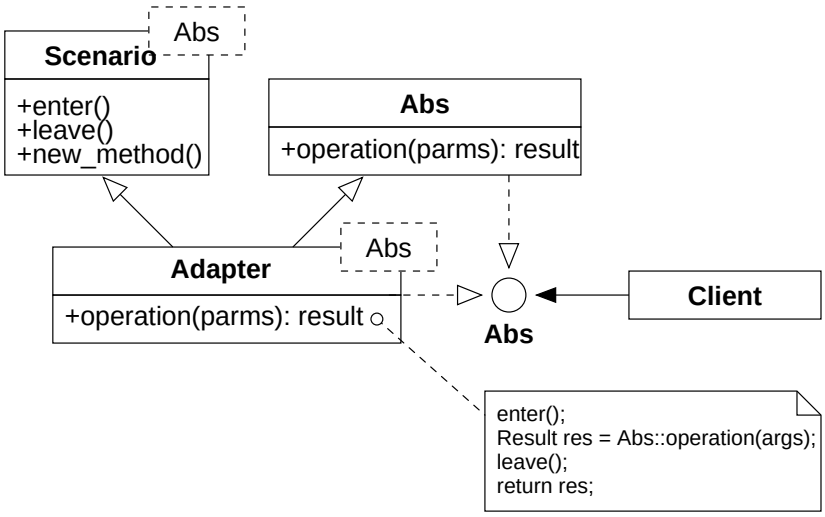


Figura 4.1: Adaptador de cenário no EPOS

Sendo assim, um adaptador de cenário para gerência de energia foi definido. A figura 4.2 mostra um diagrama UML deste adaptador de cenário. Este adaptador foi modelado para, através da extensão da interface do componente, incluir os métodos definidos pela API de gerência de energia desta proposta. Outra característica definida pela API é a abstração do retorno dos componentes a um modo operacional. A proposta é que os componentes que estiverem desligados (em um modo de operação `OFF`) sejam colocados em seu modo de operação anterior quando acessados. Para que isto ocorra o método `enter` do adaptador de cenário seria adaptado para realizar um teste antes da exe-

cução do método chamado. Contudo, foi considerado que a realização constante destes testes, que ocorreriam mesmo quando o componente estivesse em um modo operacional, aumentaria a carga de processamento do sistema de forma indesejável.

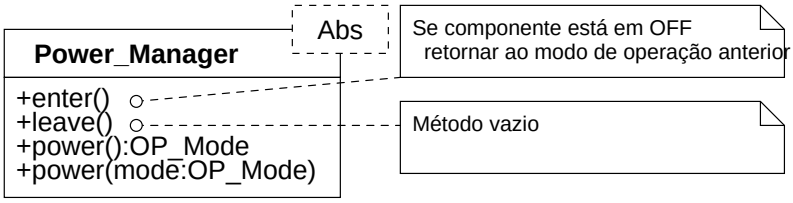


Figura 4.2: Adaptador de cenário `Power_Manager`.

Para eliminar este teste a chamada direta do método `enter` foi eliminada. A figura 4.3 apresenta um diagrama UML para esta nova solução. Nesta versão existem duas implementações para o método `enter`: `enter_on`, para ser executado quando o componente estiver em um modo funcional, e `enter_off`, para ser executado quando o componente estiver desligado. A função a ser chamada é definida pela variável `function_pointer_enter`. A função apontada por esta variável é definida no método `power`, fazendo com que apenas ocorram testes no momento das trocas de modos de operação. Esta solução não implica em custo adicional de processamento devido a novas chamadas de funções, já que o compilador utilizado otimiza o código, colocando as funções *inline*.

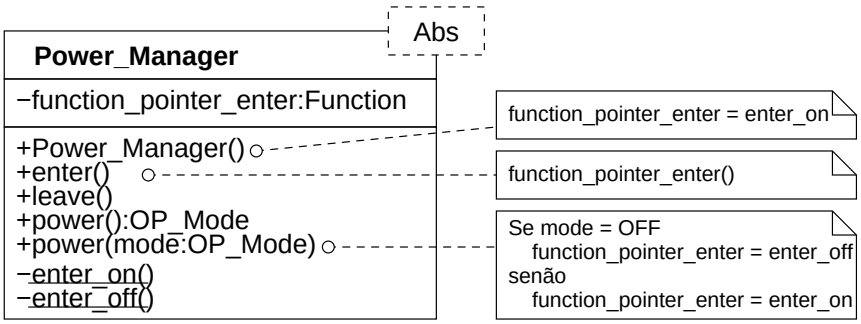


Figura 4.3: Adaptador de cenário `Power_Manager`.

Outra característica da proposta que precisa ser tratada é a geração do método `power`, que deve ser gerado com base nas redes de troca de modo de operação.

O mecanismo utilizado para extrair a implementação deste método das Redes de Petri é descrito na próxima seção.

4.3 Resolução Estática das Redes de Troca de Modos de Operação

As redes de troca de modos de operação apresentadas na seção 3.2 fornecem a esta proposta um mecanismo para especificar os procedimentos de mudança de modo de operação. Embora existam diversos modelos de análise matemática para a interpretação destas redes em tempo de execução, estas análises demandam por capacidades de processamento e memória que se encontram além dos apresentados pela maioria dos dispositivos utilizados em sistemas profundamente embarcados. Por exemplo, uma estrutura de dados para representar o grafo de uma Rede de Petri utilizaria, no mínimo, um contador para cada nó (1 byte) e referências para as conexões de cada nó (2 bytes para cada conexão em um processador com endereçamento de 16 bits), além dos dados adicionais da estrutura de controle. Por este ponto de vista simplista, cada rede generalizada (anexo A) necessitaria de 225 bytes de memória para ser representada. Neste cenário, uma rede como a gerada pelo exemplo apresentado na figura 3.5 utilizaria, grosseiramente, 900 bytes. Este consumo de memória tornaria a técnica proibitiva em plataformas como as apresentadas na seção 4.1.2, que possuem apenas alguns poucos quilobytes de memória. Além do consumo excessivo de memória, os algoritmos que manipulariam estas estruturas em tempo de execução também agregariam ao sistema custos adicionais de processamento indesejáveis.

Como ressaltado na seção 3.2, as redes de troca de modos de operação devem ser desenvolvidas de modo a não gerar situações de concorrência. Redes de Petri organizadas desta forma permitem uma execução seqüencial já que, em nenhum momento, mais de uma transição está habilitada. Isso torna possível conhecer a seqüência de disparos das transições em uma Rede de Petri. Deste modo, a interpretação em tempo de execução das redes de troca de modos de operação é desnecessária. Para substituir

esta interpretação, o código-fonte necessário para realizar as trocas de modo de operação definidas pelas redes foi extraído automaticamente.

As redes de troca de modos de operação propostas neste trabalho foram modeladas utilizando o software PIPE2 [AKH 05]. Esta ferramenta, de código aberto e desenvolvida no Colégio Imperial de Londres, além de permitir representar e simular as Redes de Petri também fornece um conjunto de ferramentas para análise das propriedades da rede desenvolvida. Esta ferramenta exporta as redes que modela em um formato chamado `PNML` (Petri Net Markup Language) [BIL 03]. Esta linguagem possui estruturas para a representação dos diferentes tipos de Redes de Petri. A `PNML` utiliza a `PNTD` (Petri Net Type Definition) para definir o tipo de rede utilizada, permitindo representar objetos comuns a todas as Redes de Petri e as extensões específicas para cada tipo. A linguagem tem sido largamente adotada para representação de vários tipos de Redes de Petri e está em vias de se tornar um padrão da ISO. A representação das redes neste formato é interessante nesta proposta pois possibilita que, no futuro, novas ferramentas venham a utiliza-las para diferentes finalidades.

Para gerar código automaticamente a partir das redes de troca de modos de operação, foi utilizado o processo apresentado pela figura 4.4. O PIPE2 foi modificado para permitir exportar o resultado da simulação da rede. Este resultado da simulação contém a seqüência em que as transições são disparadas na rede. Esta saída alimenta uma nova ferramenta, que foi chamada de *Tradutor*. O papel do Tradutor é bastante simples: ele analisa o código-fonte com a declaração do componente no sistema alvo buscando pelas declarações de outros componentes, identificando o nome das variáveis que representam estes componentes e adaptando a saída da simulação para código `C++` compilável.

Para exemplificar, a figura 4.5 mostra o resultado da simulação da rede de modos de operação que põe o componente `CMAC` no modo de operação `FULL`. De posse deste resultado, o Tradutor identifica os componentes com os quais a rede interage. Neste exemplo, os componentes identificados na rede seriam: `Radio`, `SPI` e `Timer`. Ao analisar o código fonte do componente alvo, é encontrado o trecho de código apresentado na figura 4.6. Identificadas as devidas declarações, o Tradutor produz o código-fonte

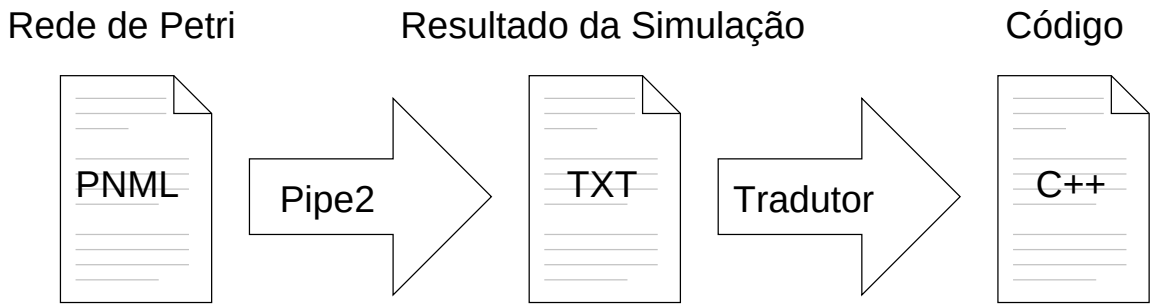


Figura 4.4: Seqüência para geração de código a partir das redes de modos de operação.

apresentado na figura 4.7.

Seguindo o modelo hierárquico das redes de modos de operação, o código das redes individuais geradas segundo o procedimento descrito anteriormente deve ser integrado à rede de modos de operação generalizada. Para isso, a rede generalizada foi submetida a um processo semelhante ao anterior para extrair código `C++` que reproduza o comportamento esperado. O problema encontrado aqui é a existência de conflito na marcação inicial da rede, isto é, na rede generalizada as transições `power(FULL)`, `power(LIGHT)`, `power(STANDBY)` e `power(OFF)` estão habilitadas simultaneamente e competindo pelo único recurso presente no lugar `Atomic_Execution`. Analisando esta rede, é observado que este é o único conflito apresentado. A partir do disparo de uma das transições em conflito, a execução se torna seqüencial até que um recurso seja novamente inserido no lugar `Atomic_Execution`. Assim sendo, as seguintes ações foram adotadas para reproduzir o comportamento desejado a partir do disparo de cada uma das transições ativadas, que é iniciado por uma chamada da aplicação:

```
1. Adquirir exclusão mútua;
2. Atualizar variáveis de modo de operação;
3. Trocar modo de operação do componente;
4. Liberar exclusão mútua;
```

A exclusão mútua foi garantida através do uso de um *mutex*. As variáveis de modo de operação são duas na implementação elaborada, uma para armazenar o

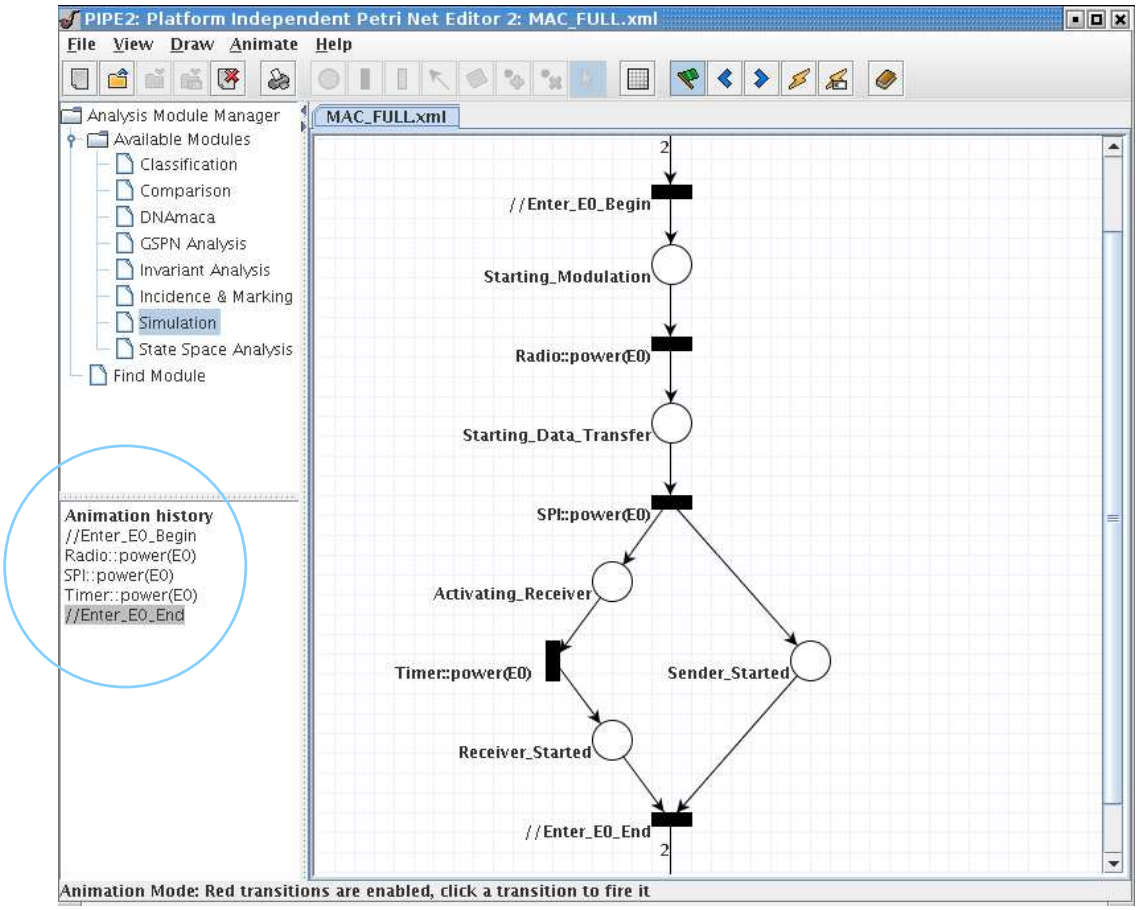


Figura 4.5: Seqüência de simulação com código sendo gerado.

```
class CMAC {

// . . .

private:
    Radio _radio;
    SPI _spi;
    Timer _timer;

//  . . .

};
```

Figura 4.6: Declarações de componentes utilizados na implementação do componente `CMAC`.

```
void power_full() {
    //Enter FULL Begin
    _radio.power(Radio::FULL);
    _spi.power(SPI::FULL);
    _timer.power(Timer::FULL);
    //Enter FULL End
}
```

Figura 4.7: Código final para a simulação da figura 4.5.

modo de operação atual, e outra para armazenar o modo de operação anterior. Esta última variável é utilizada para implementar o mecanismo que retorna ao modo de operação anterior quando um componente "desligado" é acessado. A troca de modo de operação é realizada executando o código gerado pelas redes de troca de modo de operação de cada componente. Ao final o *mutex* deve ser liberado. O conflito encontrado na marcação inicial da rede foi resolvido disparando a transição desejada baseado no parâmetro que é passado ao método `power(mode)` da API de gerência de energia. A figura 4.8 apresenta o procedimento implementado, e a figura 4.9 apresenta a nova versão do cenário `Power_Manager` para implementar esta estrutura.

4.4 Mecanismo de Propagação de Mensagens

A seção 3.3 especifica um mecanismo para permitir a interação entre os diversos componentes do sistema nos procedimentos de troca de modo de operação. Como é definido naquela seção, as mensagens são passadas através da API e a seqüência em que estas mensagens são passadas está expressa nas redes de troca de modos de operação. Sendo assim, as implementações descritas nas seções 4.2 (API) e 4.3 (redes de troca de modo de operação) já cobrem a propagação de mensagens entre os componentes que possuem algum tipo de relação (e.g., propagação da troca de modo de operação do componente `CMAC` para os componentes `Radio`, `SPI` e `Timer`). Estas estruturas, contudo, ainda não são suficientes para implementar o mecanismo de propagação para todo o sistema. Esta seção descreve a implementação da lista de instâncias de componentes que um componente global do sistema (`System`) utiliza para acessar a API de todos os

```
void Component::power(char mode) {

    _atomic_execution_mutex.lock();

    _prev_op_mode = _op_mode;
    _op_mode = mode;

    switch(mode) {

    case Component::FULL:
        power_full();
        break;

    case Component::LIGHT:
        power_light();
        break;

    case Component::STANDBY:
        power_standby();
        break;

    case Component::OFF:
        power_off();
        break;

    }

    _atomic_execution_mutex.unlock();

}
```

Figura 4.8: Procedimento de troca de modo de operação (implementação da rede generalizada).

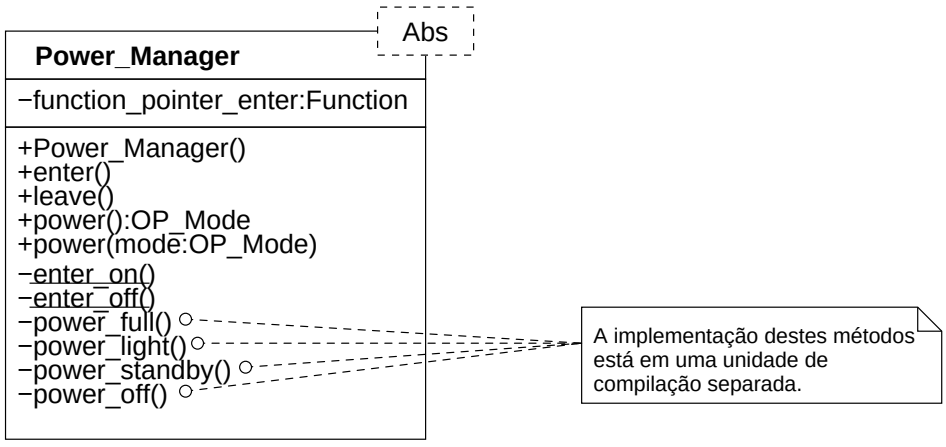


Figura 4.9: Adaptador de cenário `Power_Manager`.

componentes sendo utilizados.

Propagação Generalizada (Para Todo o Sistema)

Para implementar uma lista de instâncias de componentes do sistema é necessário, primeiramente, conhecer os componentes do sistema que estão sendo utilizados. Para isso um componente global do sistema EPOS chamado `System` foi modificado para manter uma lista de instâncias dos componentes do sistema. Esta lista utiliza uma combinação de polimorfismo e técnicas de meta-programação estática para permitir a transparência de tipos dos componentes. Feito isso, os métodos da API de gerência de energia foram adicionados ao componente. O método `power(mode:OP_Mode)` neste componente varre esta lista, propagando o modo de operação requisitado para os componentes em uso pela aplicação.

O sistema EPOS possui uma família de aspectos chamada `Shared`, que é responsável por tratar o compartilhamento de recursos do sistema. Um dos membros desta família, o aspecto `Referenced`, implementa um mecanismo de contadores de referências para os recursos do sistema. O comportamento padrão deste aspecto é o de apenas permitir que um recurso seja destruído quando não houverem mais referências a ele no sistema. Algumas abstrações, contudo, necessitam de tratamento especial para compartilhamento. Neste contexto existe, por exemplo, uma especialização deste aspecto para ser aplicada ao componente `Segment`, que abstrai segmentos de memória. Esta especialização provê suporte adequado para sistemas onde há memória compartilhada.

De modo similar, uma especialização do aspecto `Referenced` foi criada para gerência de energia. Esta especialização modifica os construtores e destrutor dos componentes para, respectivamente, guardar e descartar referências aos objetos criados na lista do componente `System`.

Compartilhamento de Recursos

A última modificação do sistema foi a inclusão de código para tratar o problema de compartilhamento de recursos descrito na seção 3.3.2. Para isto, foram adi-

cionados ao adaptador de aspecto `Referenced` especializado para gerência de energia quatro contadores, um para cada modo de operação.Nesta versão do aspecto, o construtor do componente incrementa o contador do modo de operação `FULL`. Alterações de modos de operação (chamadas ao método `power`) também são interceptadas para permitir que o componente seja mantido no modo de operação menos restrito que possuir referências. Esta interceptação também é utilizada para atualizar os contadores de modos de operação.

A implementação descrita neste capítulo focou o desenvolvimento de um conjunto de artefatos de software que, oferecendo as funcionalidades descritas no capítulo 3, não agrega ao sistema custos adicionais de processamento e memória indesejáveis. No próximo capítulo são apresentados estudos de caso do uso deste sistema, onde é feita uma análise do impacto deste em termos de custos adicionais de memória de código e dados, bem como na redução do consumo de energia.

Capítulo 5

Estudos de Caso

Duas aplicações foram desenvolvidas para testar esta proposta de gerente de energia. A primeira é um termômetro que manda leituras periódicas de temperatura através de uma porta serial. Esta aplicação é simples e foi desenvolvida para demonstrar a facilidade em tratar o consumo de energia através da API proposta. A segunda, mais complexa, utiliza uma plataforma de sensoriamento MICA2 MOTE [HIL 00]. A aplicação realiza leituras periódicas de dois sensores (temperatura e luminosidade), enviando estas leituras através do rádio.

5.1 Termômetro

Para demonstrar a usabilidade da interface definida, um termômetro serial foi implementado utilizando um protótipo com um termistor (resistor sensível a temperatura) de 10 KΩ conectado a um canal do conversor analógico-digital de um microcontrolador ATMEGA16, da Atmel [ATM 04b]. A figura 5.1 apresenta um diagrama do hardware utilizado. Este dispositivo deve operar enviando uma leitura de temperatura a cada segundo pela porta serial.

Analisando a aplicação foram identificados dois modos de operação para o sistema, aqui chamados de *ativo* e *espera*. Quando ativo, o sistema adquire uma leitura do sensor de temperatura, converte esta leitura para uma unidade real (e.g., graus

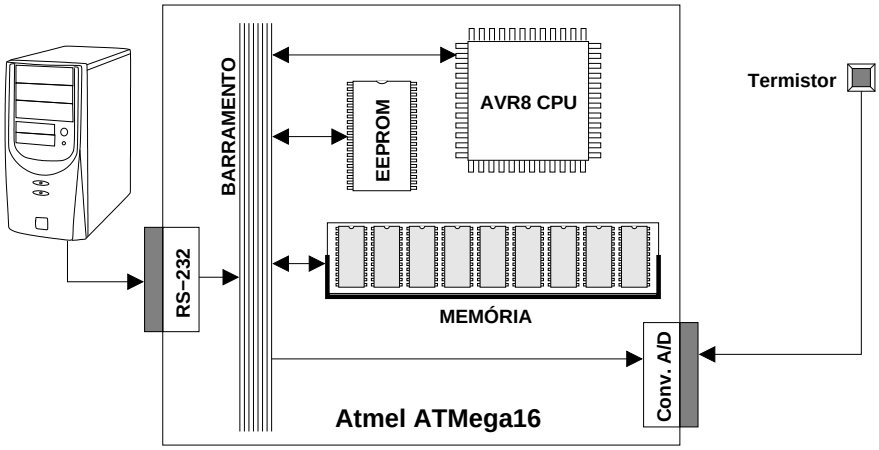


Figura 5.1: Hardware do protótipo construído.

Celcius) e envia pela porta serial. Quando o sistema está em espera, este fica aguardando que o intervalo de tempo entre leituras passe para retornar ao estado ativo. Neste último estado, o único dispositivo que precisa estar operando é o temporizador do sistema, para permitir o acionamento do estado ativo. Sendo assim, foi projetada uma aplicação que utiliza quatro componentes do EPOS: `System`, `Alarm`, `Temperature_Sensor` (membro da família de Sencientes [WAN 05]) e `UART`. Em modo ativo, todos os componentes devem estar ligados. Em modo de espera, apenas o `Alarm` deve ser mantido ligado. Assim, foram utilizados dois modos de operação: `FULL` para representar o modo *ativo* e `STANDBY` para representar o modo *espera*. No modo de operação `FULL` todos os componentes devem estar completamente operacionais. Já no modo `STANDBY`, todos os componentes, exceto o `Alarm`, devem entrar em modos restritos, de baixo consumo de energia. O componente `Alarm` deve ser mantido em um modo funcional para acordar o sistema periodicamente. A figura 5.2 apresenta o conjunto de amarrações de modos de operação realizados. Estas amarrações são as originais do sistema, ou seja, o programador da aplicação não necessita alterar as configurações do gerente de energia para esta aplicação. Caso haja a necessidade destas amarrações serem alteradas, isto pode ser feito através de uma ferramenta gráfica de configuração do EPOS.

A aplicação implementada para esta plataforma no EPOS é apresentada na figura 5.3. Quando a aplicação inicia, todos os componentes sendo utilizados são ini-

```
//Alarm
static const char FULL    = Alarm::ON;
static const char STANDBY = Alarm::ON;

//Temperature_Sensor
static const char FULL    = Temperature_Sensor::ON;
static const char STANDBY = Temperature_Sensor::STOPED;

//UART
static const char FULL    = UART::ON;
static const char STANDBY = UART::STOPED;
```

Figura 5.2: Amarrações dos modos de operação.

cializados através de seus construtores e, por definição, colocados no modo de operação `FULL`. Em seguida um evento periódico é registrado no componente `Alarm`. O modo de operação de todo o sistema é então alterado para `STANDBY` através do método `power` do componente `System`. Quando isto acontece, o componente `System` coloca todos os componentes do sistema em seus modos `STANDBY`, conforme definido pelas amarrações de modos de operação (ver figura 5.2). O `Alarm` utiliza um temporizador para gerar interrupções a uma dada frequência. A cada interrupção de tempo, a CPU, que encontra-se no modo de operação *power_save* (ver tabela 4.1.2), acorda e o componente `Alarm` trata todos os eventos registrados, executando os que atingiram seu período. Neste exemplo, a cada segundo os componentes `Temperature_Sensor` e `UART` são acordados automaticamente quando acessados e uma leitura de temperatura é enviada através da porta serial. Quando todos os eventos registrados são tratados, a aplicação continua a execução normal chegando ao laço principal, que põe o componente `System` de volta no modo de operação `STANDBY`.

Foram realizadas medições do consumo de energia para este estudo de caso. Estas leituras foram realizadas medindo a queda de tensão em um resistor de alta precisão colocado entre a fonte de alimentação e o sistema. A medição foi feita por um conversor analógico-digital operando a uma freqüência de 20 KHz e com resolução de 8 bits. Foram realizadas dez medições, cada uma com uma duração de dez segundos. Foram medidos os consumos de energia para a aplicação com e sem gerência de energia

```
#include <system.h>
#include <temperature_sensor.h>
#include <uart.h>
#include <alarm.h>

void alarm_handler() {
    static Temperature_Sensor therm;
    static UART uart;
    uart.put(therm.sample());
}

int main() {
    Handler_Function handler(&alarm_handler);
    Alarm alarm(1000000, &handler);

    while(1) {
        System::power(System::STANDBY);
    }
}
```

Figura 5.3: A aplicação Termômetro.

	Código	Dados	Energia
Sem gerenciamento	9.496 Bytes	161 Bytes	513 mJ
Com gerenciamento	10.758 Bytes	200 Bytes	472 mJ
Impacto	+11,73%	+19,5%	-8%

Tabela 5.1: Gerência de energia pelo EPOS para o estudo de caso Termômetro.

pelo sistema. A tabela 5.1 apresenta os resultados destas medições, juntamente com o impacto do gerente de energia em termos de consumo de memória para código e para dados. O ganho em consumo de energia neste estudo de caso foi relativamente pequeno (8%). Isto se deve a dois fatores: (1) o sistema foi prototipado em um *prot-o-board*, cujas conexões imperfeitas e contatos realizados por fios contribuem para gerar uma grande quantidade de ruído e (2) o microcontrolador utilizado não foi projetado para aplicações onde o consumo de energia é crítico. O próximo estudo de caso utiliza um outro microcontrolador da mesma família cujas características permitem um melhor controle do consumo de energia.

5.2 Módulo de Sensoriamento *Mica Mote2*

O segundo estudo de caso realizado para testar a proposta foi desenvolvido sobre a plataforma de redes de sensores sem fio descrita na seção 4.1.2. Um diagrama do hardware desta plataforma é apresentada na figura 5.4. Neste estudo de caso o dispositivo deve operar enviando através do rádio leituras de temperatura a cada segundo e de luminosidade a cada cem miléssimos de segundo. Neste exemplo há compartilhamento do conversor analógico-digital (ADC) pelas abstrações dos sensores de temperatura e luminosidade, o que torna necessário a utilização do mecanismo de controle de compartilhamento de recursos descrito na seção 4.4.

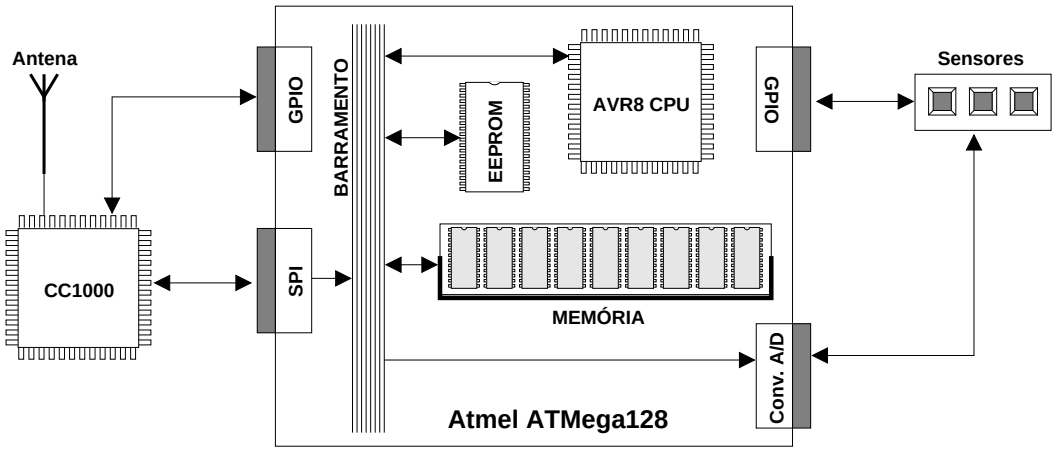


Figura 5.4: Diagrama de hardware do *Mica2 Mote*.

A aplicação implementada para esta plataforma no EPOS é apresentada na figura 5.5. Quando a aplicação inicia, todos os componentes sendo utilizados são inicializados através de seus construtores e, por definição, colocados no modo de operação `FULL`. Em seguida dois eventos periódicos são registrados no componente `Alarm`, um para enviar leituras de temperatura a cada segundo e outro para enviar leituras de luminosidade a cada cem milissegundos. O modo de operação de todo o sistema é então alterado para `STANDBY` através do método `power` do componente `System`. Quando isto acontece, o componente `System` coloca todos os componentes do sistema no modo `STANDBY`, conforme definido pelas amarrações de modos de operação (ver figura 5.6).

```
#include <system.h>
#include <nic.h>
#include <sensor.h>
#include <alarm.h>

NIC radio;

void temperature() {
    static Temperature_Sensor temperature;
    int buffer = temperature.sample();
    temperature.power(Temperature_Sensor::OFF);
    radio.send(address, protocol, &buffer, sizeof(int));
    radio.power(NIC::OFF);
}

void photo() {
    static Photo_Sensor photo;
    int  buffer = photo.sample();
    photo.power(Photo_Sensor::OFF);
    radio.send(address, protocol, &buffer, sizeof(int));
    radio.power(NIC::OFF);
}

int main() {
    Handler_Function temperature_handler(&temperature);
    Alarm temperature_alarm(1000000, &temperature_handler);

    Handler_Function photo_handler(&photo);
    Alarm photo_alarm(100000, &photo_handler);

    while(1) {
        System::power(System::STANDBY);
    }
}
```

Figura 5.5: Aplicação para o *Mica2 Mote*.

O Alarm utiliza um temporizador para gerar interrupções a uma dada freqüência. A cada interrupção de tempo, a CPU acorda e o componente Alarm trata todos os eventos registrados, executando os que atingiram seu período. Neste exemplo, a cada cem milissegundos o componente Photo_Sensor é acordado automaticamente quando acessado e uma leitura de luminosidade é realizada. Após esta leitura, o sensor é desligado e o rádio é acordado automaticamente quando o envio é realizado. A mesma sequência de eventos ocorre para a leitura de temperatura através do componente Temperature_Sensor porém, neste caso, com período de um segundo. Quando todos os eventos registrados são tratados, a aplicação continua a execução normal chegando ao laço principal, que põe o componente System de volta no modo de operação STANDBY.

```
//Alarm
static const char FULL    = Alarm::ON;
static const char STANDBY = Alarm::ON;

//Temperature_Sensor
static const char FULL    = Temperature_Sensor::ON;
static const char STANDBY = Temperature_Sensor::STOPED;

//Light_Sensor
static const char FULL    = Light_Sensor::ON;
static const char STANDBY = Light_Sensor::STOPED;

//CMAC
static const char FULL    = CMAC::ON;
static const char STANDBY = CMAC::STOPED;
```

Figura 5.6: Amarrações dos modos de operação.

A cada segundo nesta aplicação o sistema realiza leituras nos dois sensores. Nestes pontos há concorrência no acesso ao ADC, conforme apresentado na figura 3.7, neste caso com sensores de luminosidade e temperatura. Nesta situação o mecanismo de contadores implementado pelo aspecto Shared faz com que o ADC permaneça ligado até que ambos os sensores (Photo_Sensor e Temperature_Sensor) solicitem seu desligamento.

Foram realizadas medições do consumo de energia para este estudo de caso. Estas leituras foram realizadas medindo a queda de tensão em um resistor de alta

precisão colocado entre a fonte de alimentação e o sistema. A medição foi feita por um conversor analógico-digital operando a uma frequência de 20 KHz e com resolução de 8 bits. Foram realizadas dez medições, cada uma com uma duração de dez segundos. Foram medidos os consumos de energia para a aplicação com e sem gerenciamento de energia pelo sistema. A tabela 5.2 apresenta os resultados destas medições, juntamente com o impacto do gerente de energia em termos de consumo de memória para código e para dados. Neste caso, a economia de energia reflete, principalmente, a redução no consumo do processador e dos sensores quando estes estão em modos de mais baixo consumo. Isto ocorre porque a implementação do `CMAC` já utiliza o rádio de forma a minimizar o consumo de energia, desligando partes não utilizadas quando desnecessário (e.g., ligando o circuito de envio apenas quando há dados para serem enviados). Optou-se por utilizar a mesma implementação em ambos os testes por considerar justo, já que seria ilógico forçar o rádio a consumir mais energia sem que haja necessidade para isso.

	Código	Dados	Energia
`Sem gerenciamento`	12.900 Bytes	294 Bytes	73,6 mJ
`Com gerenciamento`	14.294 Bytes	345 Bytes	60,8 mJ
`Impacto`	+9,7%	+14,8%	-17,5%

Tabela 5.2: Gerência de energia pelo EPOS para o estudo de caso *Mica2 Mote*.

5.3 Discussão

Os estudos de caso apresentados neste capítulo tinham como objetivo mostrar como a infra-estrutura para gerência de energia definida neste trabalho é utilizada. Estes estudos de caso, contudo, são relativamente simples. Isto ocorre porque os protótipos foram desenvolvidos para sistemas de sensoriamento, cujas aplicações são frequentemente simples. Estas aplicações, porém, também se beneficiam do uso do mecanismo de gerência de energia definido neste trabalho. A importância da minimização

do consumo de energia nestes sistemas é latente, dada a naturaza de sua implantação, que pode envolver locais de difícil acesso, inviabilizando a troca de baterias. Nestas situações a vida útil das baterias é a vida útil do sistema.

Aplicações mais complexas também se beneficiariam desta técnica ao passo que utilizem mais componentes, aumentando o número de relações e dependências entre estes componentes; passem a utilizar múltiplas *threads*, aumentando o compartilhamento de recursos do sistema; e aumentem a carga de trabalho do sistema, passando a consumir mais energia. O aumento desta complexidade, contudo, pode também trazer problemas à gerência de energia. Dentre estes problemas está o fato de a concorrência pelo uso de certos dispositivos vir a gerar alterações muito frequentes de modos de operação, o que pode acabar aumentando o consumo de energia do sistema. Outra possível falha do sistema nestas situações é a confiança nas tarefas. Em um ambiente multi-tarefa, se uma das tarefas do sistema não implementar gerência de energia existe o risco de componentes serem mantidos ligados, mesmo quando ociosos, devido ao mecanismo de contadores implementado para controle de concorrência. Nestas situações, um gerente de energia ativo, que verifique constantemente o estado de cada componente do sistema, provavelmente economizaria mais energia.

Mesmo sendo simples, as aplicações utilizadas nos estudos de caso permitiram demonstrar o uso da interface. O objetivo deste trabalho não é desenvolver um gerente de energia mais eficiente que outros já implementados. O objetivo deste trabalho é desenvolver uma infra-estrutura que permita a integração de mecanismos de gerência de energia diversos sob uma única interface, que possa ser facilmente utilizada por um programador sem que haja necessidade deste conhecer detalhes do hardware que está sendo utilizado. Por este motivo, não foram realizados testes comparativos com as técnicas apresentadas no capítulo 2. Em termos de consumo de energia, aquelas técnicas seriam mais eficientes que as implementadas aqui.

Em termos de desempenho, o impacto da estrutura desenvolvida no tempo de execução das aplicações é bastante reduzido. Isto ocorre devido ao uso intensivo de técnicas de metaprogramação estática, os templates em C++, cuja resolução em tempo de compilação permite diluição do código de gerência de energia nos compo-

nentes alvo, eliminando, principalmente, chamadas de função, que impactariam de modo bastante negativo no tempo de execução. Atrasos significativos, contudo, podem ocorrer devido a características dos dispositivos utilizados. Por exemplo, ao ligar um termistor (permitindo passagem de corrente através dele) é necessário aguardar um tempo (e.g., um milisegundo no exemplo da seção 5.2) para que a corrente estabilize e seja possível obter leituras corretas do dispositivo. Tempos de estabilização ainda maiores são necessários para alguns outros dispositivos, principalmente aqueles que possuem partes mecânicas, como discos para iniciar a rotação do cilindro. Nestes casos, a perda de prazos na execução de tarefas pode se tornar um problema crítico, o que é, de fato, foco de muita pesquisa hoje.

Embora este trabalho não tenha implementado técnicas complexas como as muitas apresentadas no capítulo 2, ele não impede que estas técnicas sejam integradas a este sistema. Dada a configurabilidade disponibilizada pela API de gerência de energia desenvolvida aqui, é possível integrar qualquer um daqueles trabalhos ao sistema desenvolvido. Por exemplo, uma CPU com suporte a DVS contém modos de operação distindos para cada freqüência de operação que suporte. Assim, diferentes heurísticas para aplicação de DVS poderiam ser implementadas como diferentes modos de operação do componente escalonador, que gerenciaria os modos de operação da CPU. Outro exemplo seria a implementação de técnicas para gerenciar o consumo de energia de dispositivos de armazenamento de dados, como discos ou memórias persistentes. Um dos modos de operação de um sistema de arquivos poderia manter os dispositivos desligados, agrupando requisições de leitura e escrita até que, ou hajam muitas tarefas bloqueadas no sistema, ou o volume de requisições seja suficiente para justificar o ligamento do dispositivo.

Capítulo 6

Conclusão

O termo *power-aware* está cada dia mais presente nas discussões e projetos sobre sistemas computacionais. De fato, este é um problema que precisa ser tratado em sistemas de todos os tipos, desde grandes centros de computação (e.g., agregados e grades computacionais) até pequenos dispositivos de computação embarcada e dedicada que são, freqüentemente, alimentados por bateria. Este trabalho concentrou seus estudos nas técnicas para gerenciar o consumo de energia destes últimos sistemas. Nestes estudos foi possível observar que as melhores técnicas de gerência de energia são sempre aquelas que, de algum modo, consideram características e comportamento das aplicações como diretrizes para a gerência de energia. Assim sendo, foi considerada uma boa estratégia entregar a gerência de energia do sistema à aplicação.

Ao entregar o controle do consumo de energia para a aplicação, contudo, surge uma nova série de problemas que precisam ser resolvidos. Um destes problemas é o estabelecimento de um mecanismo pelo qual a aplicação possa interagir com o sistema operacional e, assim, agir como gerente de energia deste sistema. Buscando agregar o mínimo possível de complexidade ao processo de desenvolvimento de aplicações embarcadas, foi definida uma API configurável de gerência de energia através da qual os modos de operação dos componentes do sistema podem ser consultados ou alterados. Nesta proposta é esperado que o programador da aplicação comunique ao sistema quando um determinado componente passará por um período de inatividade, informando que este

componente pode ser desligado. A API garante que estes componentes voltem aos seus modos de operação anteriores automaticamente quando utilizados.

Este mecanismo de gerência de energia foi concebido para operar com sistemas operacionais baseados em componentes. Nestes sistemas componentes são organizados para separar funcionalidades do sistema operacional, facilitando a identificação dos componentes que podem ser desligados em determinado momento por parte do programador da aplicação. Em sistemas baseados em componentes, contudo, componentes costumam interagir para implementar as funcionalidades desejadas. Por exemplo, um componente de comunicação agrega diversos componentes que implementam diferentes protocolos (e.g., `TCP`, `IP`), os drivers de dispositivo (e.g., `Ethernet_NIC`, `Radio_NIC`), etc. De modo a abstrair as relações entre componentes para gerenciar consumo de energia, este trabalho propôs um mecanismo baseado em redes de Petri que formaliza estas relações durante trocas de modos de operação. A análise matemática destas redes mostra que o sistema de gerência de energia proposto é livre de impasse e, através de análise do grafo de alcançabilidade desta rede, que todos os estados desejados são alcançáveis, e que estados indesejados não o são.

Considerando a gerência de energia uma característica não-funcional de sistemas computacionais e, portanto, fatorável como um aspecto [LOH 05], este gerente foi modelado como tal. O sistema operacional EPOS, que oferece suporte para utilização de aspectos, foi utilizado para gerar os protótipos desta proposta. Das redes de troca de modos de operação foi extraído o código-fonte necessário para que as trocas realmente ocorram. Isto foi feito combinando uma ferramenta de simulação de redes de Petri e uma ferramenta desenvolvida neste trabalho que, analisando o sistema alvo (EPOS), traduz os resultados da simulação para código `C++` compilável naquele sistema. O código-fonte gerado é agrupado em um módulo de compilação separado e ligado ao sistema quando este é gerado.

Estudos de caso neste trabalho demonstraram como a API pôde ser configurada para satisfazer as necessidades específicas de cada aplicação. Estes estudos de caso ainda mostram que o papel do programador da aplicação como “gerente de energia” do sistema não deve tornar a tarefa de desenvolvimento desta aplicação mais complexa

já que, a partir de uma análise da aplicação, é possível identificar os momentos em que cada componente deve estar em cada modo de operação, não sendo complicado inserir no código da aplicação chamadas à API de gerência de energia. Mecanismos especificados nesta proposta ainda liberam o programador da aplicação de se preocupar com concorrência no acesso a componentes (e.g., duas *threads* utilizando o mesmo dispositivo de comunicação), e a integração dos componentes através das redes de modo de operação permite ao programador trocar os modos de operação de todos os componentes do sistema em conjunto.

É fato, porém, que esta proposta possui limitações. Uma das limitações existentes é o emprego das redes de troca de modos de operação para gerar código-fonte de gerência de energia. Como este mecanismo de gerência de energia foi implementado como um aspecto e, portanto, não está diretamente integrado ao sistema, mudanças no sistema alvo podem fazer com que o aspecto de gerência de energia ou deixe de operar ou passe a não ser totalmente eficiente. Algumas mudanças no sistema alvo podem requerer mudanças nas redes de troca de modos de operação e, por consequência, a re-geração de código para o sistema. Outra limitação, não tão grave, é o fato de tornar a gerência de energia do sistema alvo dependente de uma ferramenta externa ao sistema. Esta limitação fica menos importante ao passo que, com o tempo, as redes de troca de modos de operação tomem forma estável. A partir daí seria possível manter unidades de compilação prontas, não sendo necessário o uso das ferramentas de gerência de energia a cada compilação do sistema.

Ainda como limitação desta proposta cabe citar o fato de que, assim como gerentes de energia convencionais que tomam decisões em tempo de execução, o programador da aplicação também está propenso a cometer erros no gerenciamento de energia. O principal erro que um programador de aplicação pode gerar é o de ligar e desligar dispositivos muito freqüentemente. A maioria dos dispositivos consomem para ligar ou desligar uma quantidade de energia comparável à sua operação por um determinado período de tempo. Em alguns casos é interessante manter os dispositivos ligados caso estes devam estar novamente ativos dentro de um curto período de tempo. Identificar estes casos é, contudo, uma tarefa difícil para o programador, já que atividades do sis-

tema podem estar condicionadas a eventos esporádicos, cuja freqüência não se conhece. Um trabalho em andamento está estudando os custos de ligar e desligar componentes para estender este mecanismo de gerência de energia possibilitando que, através de uma análise do comportamento dos dispositivos individualmente, seja possível impedir que o programador da aplicação cometa estes erros.

Outro trabalho em andamento está utilizando a infra-estrutura de gerência de energia definida nesta proposta para computar o consumo de energia de tarefas imprecisas para guiar o processo de tomada de decisão quanto à execução ou não dos trechos imprecisos destas tarefas. No futuro este mecanismo deve ser integrado a um sistema de QoS (*Quality of Service*) que permitirá o uso de consumo de energia como parâmetro de QoS dos sistemas computacionais.

Referências Bibliográficas

[ABO 03] ABOUGHAZALEH, N. et al. Energy management for real-time embedded applications with compiler support. In: LCTES '03: PROCEEDINGS OF THE 2003 ACM SIGPLAN CONFERENCE ON LANGUAGE, COMPILER, AND TOOL FOR EMBEDDED SYSTEMS, 2003. **Proceedings...** New York, NY, USA: ACM Press, 2003. p.284–293.

[AKH 05] AKHARWARE, N. **PIPE2: Platform Independent Petri Net Editor**. London: Imperial College of Science, Technology and Medicine, 2005. Dissertação de Mestrado.

[ANA 04] ANAND, M.; NIGHTINGALE, E. B.; FLINN, J. Ghosts in the machine: Interfaces for better power management. In: PROCEEDINGS OF THE SECOND INTERNATIONAL CONFERENCE ON MOBILE SYSTEMS, APPLICATIONS, AND SERVICES (MOBISYS'04), 2004. **Proceedings...** Boston, USA: [s.n.], 2004.

[AS 04] AS, C. **SmartRF CC1000 Datasheet**. Oslo, Norway, 2.2. ed., Apr, 2004.

[ATM 04a] ATMEL. **ATMega128L Datasheet**. San Jose, CA, 2467M. ed., Nov, 2004.

[ATM 04b] ATMEL. **ATMega16L Datasheet**. San Jose, CA, 2466J. ed., Oct, 2004.

[AZE 02] AZEVEDO, A. et al. Profile-based dynamic voltage scheduling using program checkpoints. In: DATE '02: PROCEEDINGS OF THE CONFERENCE ON DESIGN, AUTOMATION AND TEST IN EUROPE, 2002. **Proceedings...** Washington, DC, USA: IEEE Computer Society, 2002. p.168.

[BEN 98] BENINI, L.; BOGLIOLO, A.; MICHELI, G. D. Dynamic power management of electronic systems. In: ICCAD '98: PROCEEDINGS OF THE 1998 IEEE/ACM INTERNATIONAL CONFERENCE ON COMPUTER-AIDED DESIGN, 1998. **Proceedings...** New York, NY, USA: ACM Press, 1998. p.696–702.

[BIL 03] BILLINGTON, J. et al. The petri net markup language: Concepts, technology, and tools. In: APPLICATIONS AND THEORY OF PETRI NETS 2003: 24TH INTERNATIONAL CONFERENCE, ICATPN 2003, 2003. **Proceedings...** Eindhoven, The Netherlands: Springer, 2003. v.2679/2003 of **LNCS**, p.483–505.

[CHA 92] CHANDRAKASAN, A. P.; SHENG, S.; BRODERSEN, R. W. Low-power cmos digital design. **IEEE Journal of Solid-State Circuits**, [S.l.], v.27, n.4, p.473–484, Apr, 1992.

[CHE 04] CHEN, D. et al. Low-power technology mapping for fpga architectures with dual supply voltages. In: FPGA '04: PROCEEDINGS OF THE 2004 ACM/SIGDA 12TH INTERNATIONAL SYMPOSIUM ON FIELD PROGRAMMABLE GATE ARRAYS, 2004. **Proceedings...** New York, NY, USA: ACM Press, 2004. p.109–117.

[CRO 06] CROSSBOW. **MPR-MIB Users Manual**. San Jose, USA, Rev. B. ed., 2006.

[CUL 01] CULLER, D. E. et al. A network-centric approach to embedded software for tiny devices. In: EMSOFT, 2001. **Proceedings...** Tahoe City, CA, USA: Springer, 2001. v.2211 of **Lecture Notes in Computer Science**.

[CZA 98] CZARNECKI, K. et al. Generative Programming and Active Libraries. In: REPORT OF THE DAGSTUHL SEMINAR ON GENERIC PROGRAMMING, 1998. **Proceedings...** Schloß Dagstuhl, Germany: [s.n.], 1998.

[D'A 05] D'AGOSTINI, T. S.; FRöHLICH, A. A. Bridging AOP to SMP: turning GCC into a metalanguage preprocessor. In: 20TH ANNUAL ACM SYMPOSIUM ON APPLIED COMPUTING, 2005. **Proceedings...** Santa Fe, U.S.A.: [s.n.], 2005. p.1563–1564.

[EBE 04] EBERGEN, J.; GAINSLEY, J.; CUNNINGHAM, P. Transistor sizing: how to control the speed and energy consumption of a circuit. In: PROCEEDINGS OF THE 10TH INTERNATIONAL SYMPOSIUM ON ASYNCHRONOUS CIRCUITS AND SYSTEMS, 2004. **Proceedings...** Crete, Greece: IEEE, 2004. p.51–61.

[ELL 99] ELLIS, C. S. The case for higher-level power management. In: HOTOS '99: PROCEEDINGS OF THE THE SEVENTH WORKSHOP ON HOT TOPICS IN OPERATING SYSTEMS, 1999. **Proceedings...** Washington, DC, USA: IEEE Computer Society, 1999. p.162.

[ERN 03] ERNST, D. et al. Razor: A low-power pipeline based on circuit-level timing speculation. In: MICRO 36: PROCEEDINGS OF THE 36TH ANNUAL IEEE/ACM INTERNATIONAL SYMPOSIUM ON MICROARCHITECTURE, 2003. **Proceedings...** Washington, DC, USA: IEEE Computer Society, 2003. p.7.

[FLA 01] FLAUTNER, K.; REINHARDT, S.; MUDGE, T. Automatic performance setting for dynamic voltage scaling. In: MOBICOM '01: PROCEEDINGS OF THE 7TH ANNUAL INTERNATIONAL CONFERENCE ON MOBILE COMPUTING AND NETWORKING, 2001. **Proceedings...** New York, NY, USA: ACM Press, 2001. p.260–271.

[FLI 99] FLINN, J.; SATYANARAYANAN, M. Energy-aware adaptation for mobile applications. In: SOSP '99: PROCEEDINGS OF THE SEVENTEENTH ACM SYMPOSIUM ON OPERATING SYSTEMS PRINCIPLES, 1999. **Proceedings...** New York, NY, USA: ACM Press, 1999. p.48–63.

[FOR 97] FORD, B. et al. The Flux OS Toolkit: Reusable Components for OS Implementation. In: PROCEEDINGS OF THE SIXTH WORKSHOP ON HOT TOPICS IN OPERATING SYSTEMS, 1997. **Proceedings...** Cape Cod, U.S.A.: [s.n.], 1997. p.14–19.

[FRö 01] FRöHLICH, A. A. **Application-Oriented Operating Systems**. Sankt Augustin: GMD - Forschungszentrum Informationstechnik, 2001. 200 p.

[GOV 95] GOVIL, K.; CHAN, E.; WASSERMAN, H. Comparing algorithm for dynamic speed-setting of a low-power cpu. In: MOBICOM '95: PROCEEDINGS OF THE 1ST ANNUAL INTERNATIONAL CONFERENCE ON MOBILE COMPUTING AND NETWORKING, 1995. **Proceedings...** New York, NY, USA: ACM Press, 1995. p.13–25.

[GRO 03] GROVER, A. Modern system power management. **ACM Queue**, New York, NY, USA, v.1, n.7, p.66–72, 2003.

[GRU 01] GRUIAN, F. Hard real-time scheduling for low-energy using stochastic data and dvs processors. In: ISLPED '01: PROCEEDINGS OF THE 2001 INTERNATIONAL SYMPOSIUM ON LOW POWER ELECTRONICS AND DESIGN, 2001. **Proceedings...** New York, NY, USA: ACM Press, 2001. p.46–51.

[HEA 04] HEATH, T. et al. Code transformations for energy-efficient device management. **IEEE Transactions on Computers**, [S.l.], v.53, n.8, August, 2004.

[HIL 00] HILL, J. et al. System architecture directions for networked sensors. In: PROCEEDINGS OF THE NINTH INTERNATIONAL CONFERENCE ON ARCHITECTURAL SUPPORT FOR PROGRAMMING LANGUAGES AND OPERATING SYSTEMS, 2000. **Proceedings...** Cambridge, Massachusetts, United States: [s.n.], 2000. p.93–104.

[HP 04] HEWLETT-PACKARD, C. et al. **Advanced Configuration and Power Interface Specification**, 3.0. ed., Sep, 2004.

[HSU 03] HSU, C.-H.; KREMER, U. The design, implementation, and evaluation of a compiler algorithm for cpu energy reduction. In: PLDI '03: PROCEEDINGS OF THE ACM SIGPLAN 2003 CONFERENCE ON PROGRAMMING LANGUAGE DESIGN AND IMPLEMENTATION, 2003. **Proceedings...** New York, NY, USA: ACM Press, 2003. p.38–48.

[INT 96] INTEL, C.; MICROSOFT, C. **Advanced Power Management (APM) BIOS Interface Specification**, 1.2. ed., Feb, 1996.

[KEL 06] KELLNER, S. **Energy Accounting and Control for Sensor Nodes**. Erlangen, Germany: University of Erlangen-Nürnberg, jan, 2006. Diplomarbeit.

[KIC 97] KICZALES, G. et al. Aspect-Oriented Programming. In: PROCEEDINGS OF THE EUROPEAN CONFERENCE ON OBJECT-ORIENTED PROGRAMMING'97, 1997. **Proceedings...** Jyväskylä, Finland: Springer, 1997. v.1241 of **Lecture Notes in Computer Science**, p.220–242.

[KUR 04] KURSUN, E.; GHIASI, S.; SARRAFZADEH, M. Transistor level budgeting for power optimization. In: ISQED '04: PROCEEDINGS OF THE 5TH INTERNATIONAL SYMPOSIUM ON QUALITY ELECTRONIC DESIGN, 2004. **Proceedings...** Washington, DC, USA: IEEE Computer Society, 2004. p.116–121.

[LEE 00] LEE, K. Ieee 1451: A standard in support of smart transducer networking. In: PROCEEDINGS OF THE IEEE INSTRUMENTATION AND MEASUREMENT TECHNOLOGY CONFERENCE, 2000. **Proceedings...** Baltimore, MD: [s.n.], 2000. p.525–528.

[LI 04] LI, H.; KATKOORI, S.; MAK, W.-K. Power minimization algorithms for lut-based fpga technology mapping. **ACM Trans. Des. Autom. Electron. Syst.**, New York, NY, USA, v.9, n.1, p.33–51, 2004.

[LOH 05] LOHMANN, D.; SCHRöDER-PREIKSCHAT, W.; SPINCZYK, O. Functional and non-functional properties in a family of embedded operating systems. In: PROCEEDINGS OF THE TENTH IEEE INTERNATIONAL WORKSHOP ON OBJECT-ORIENTED REAL-TIME DEPENDABLE SYSTEMS, 2005. **Proceedings...** Sedona, USA: IEEEPress, 2005.

[LOR 01] LORCH, J. R.; SMITH, A. J. Improving dynamic voltage scaling algorithms with pace. In: SIGMETRICS '01: PROCEEDINGS OF THE 2001 ACM SIGMETRICS INTERNATIONAL CONFERENCE ON MEASUREMENT AND MODELING OF COMPUTER SYSTEMS, 2001. **Proceedings...** New York, NY, USA: ACM Press, 2001. p.50–61.

[MOH 05] MOHAPATRA, S. et al. A cross-layer approach for power-performance optimization in distributed mobile systems. In: IPDPS '05: PROCEEDINGS OF THE 19TH IEEE INTERNATIONAL PARALLEL AND DISTRIBUTED PROCESSING SYMPOSIUM (IPDPS'05) - WORKSHOP 10, 2005. **Proceedings...** Washington, DC, USA: IEEE Computer Society, 2005. p.218.1.

[PAN 04] PANASONIC. **ERTJ Multilayer Chip NTC Thermistors Datasheet**. Panasonic, 2004.

[PEN 02] PENZES, P.; NYSTROEM, M.; MARTIN, A. Transistor sizing of energy-delay–efficient circuits. California Institute of Technology, Apr, 2002. Relatório Técnico318.

[PER 02] PEREIRA, C.; GUPTA, R.; SRIVASTAVA, M. Pasa: A software architecture for building power aware embedded systems. In: PROCEEDINGS OF THE 2002 IEEE CAS WORKSHOP ON WIRELESS COMMUNICATION AND NETWORKING, 2002. **Proceedings...** Pasadena, USA: [s.n.], 2002.

[PET 77] PETERSON, J. L. Petri nets. **ACM Comput. Surv.**, New York, NY, USA, v.9, n.3, p.223–252, 1977.

[POL 99] POLLACK, F. J. New microarchitecture challenges in the coming generations of cmos process technologies (keynote address)(abstract only). In: MICRO 32: PROCEEDINGS OF THE 32ND ANNUAL ACM/IEEE INTERNATIONAL SYMPOSIUM ON MICROARCHITECTURE, 1999. **Proceedings...** Washington, DC, USA: IEEE Computer Society, 1999. p.2.

[POL 04] POLPETA, F. V.; FRöHLICH, A. A. Hardware Mediators: a Portability Artifact for Component-Based Systems. In: INTERNATIONAL CONFERENCE ON EMBEDDED AND UBIQUITOUS COMPUTING, 2004. **Proceedings...** Aizu, Japan: Springer, 2004. v.3207 of **Lecture Notes in Computer Science**, p.271–280.

[RUT 01] RUTENBAR, R. A. et al. Low-power technology mapping for mixed-swing logic. In: ISLPED '01: PROCEEDINGS OF THE 2001 INTERNATIONAL SYMPOSIUM ON LOW POWER ELECTRONICS AND DESIGN, 2001. **Proceedings...** New York, NY, USA: ACM Press, 2001. p.291–294.

[SAC 03] SACHS, D. G.; ADVE, S. V.; JONES, D. L. Cross-layer adaptive video coding to reduce energy on general-purpose processors. In: PROCEEDINGS OF THE 2003 INTERNATIONAL CONFERENCE ON IMAGE PROCESSING, 2003. **Proceedings...** Barcelona, Spain: [s.n.], 2003. v.3, p.109–112.

[SAC 06] SACHS, D. G. **A new framework for hierarchical cross-layer adaptation**. Urbana, USA: University of Illinois at Urbana-Champaign, 2006. Tese de Doutorado.

[SHI 01] SHIN, D.; KIM, J.; LEE, S. Low-energy intra-task voltage scheduling using static timing analysis. In: DAC '01: PROCEEDINGS OF THE 38TH CONFERENCE ON DESIGN AUTOMATION, 2001. **Proceedings...** New York, NY, USA: ACM Press, 2001. p.438–443.

[SMA 95] SMALL, C.; SELTZER, M. Structuring the Kernel as a Toolkit of Extensible, Reusable Components. In: PROCEEDINGS OF THE 1995 INTERNATIONAL WORKSHOP ON

OBJECT ORIENTATION IN OPERATING SYSTEMS, 1995. **Proceedings...** Lund, Sweden: [s.n.], 1995. p.134–137.

[SUL 04] SULTANIA, A. K.; SYLVESTER, D.; SAPATNEKAR, S. S. Transistor and pin reordering for gate oxide leakage reduction in dual tox circuits. In: ICCD '04: PROCEEDINGS OF THE IEEE INTERNATIONAL CONFERENCE ON COMPUTER DESIGN (ICCD'04), 2004. **Proceedings...** Washington, DC, USA: IEEE Computer Society, 2004. p.228–233.

[TAN 03] TAN, T. K.; RAGHUNATHAN, A.; JHA, N. K. Software architectural transformations: A new approach to low energy embedded software. In: DATE '03: PROCEEDINGS OF THE CONFERENCE ON DESIGN, AUTOMATION AND TEST IN EUROPE, 2003. **Proceedings...** Washington, DC, USA: IEEE Computer Society, 2003. p.11046.

[VEN 05] VENKATACHALAM, V.; FRANZ, M. Power reduction techniques for microprocessor systems. **ACM Comput. Surv.**, New York, NY, USA, v.37, n.3, p.195–237, 2005.

[WAN 05] WANNER, L. F. et al. Operating System Support for Handling Heterogeneity in Wireless Sensor Networks. In: 10TH IEEE INTERNATIONAL CONFERENCE ON EMERGING TECHNOLOGIES AND FACTORY AUTOMATION, 2005. **Proceedings...** Catania, Italy: [s.n.], 2005.

[WAN 06] WANNER, L. F. **Suporte de sistema operacional para rede de sensores sem fio**. Florianpolis: Federal University of Santa Catarina, 2006. Dissertação de Mestrado.

[WEI 94] WEISER, M. et al. Scheduling for reduced cpu energy. In: PROCEEDINGS OF THE FIRST SYMPOSIUM ON OPERATING SYSTEMS DESIGN AND IMPLEMENTATION, 1994. **Proceedings...** Monterey, USA: [s.n.], 1994. p.13–23.

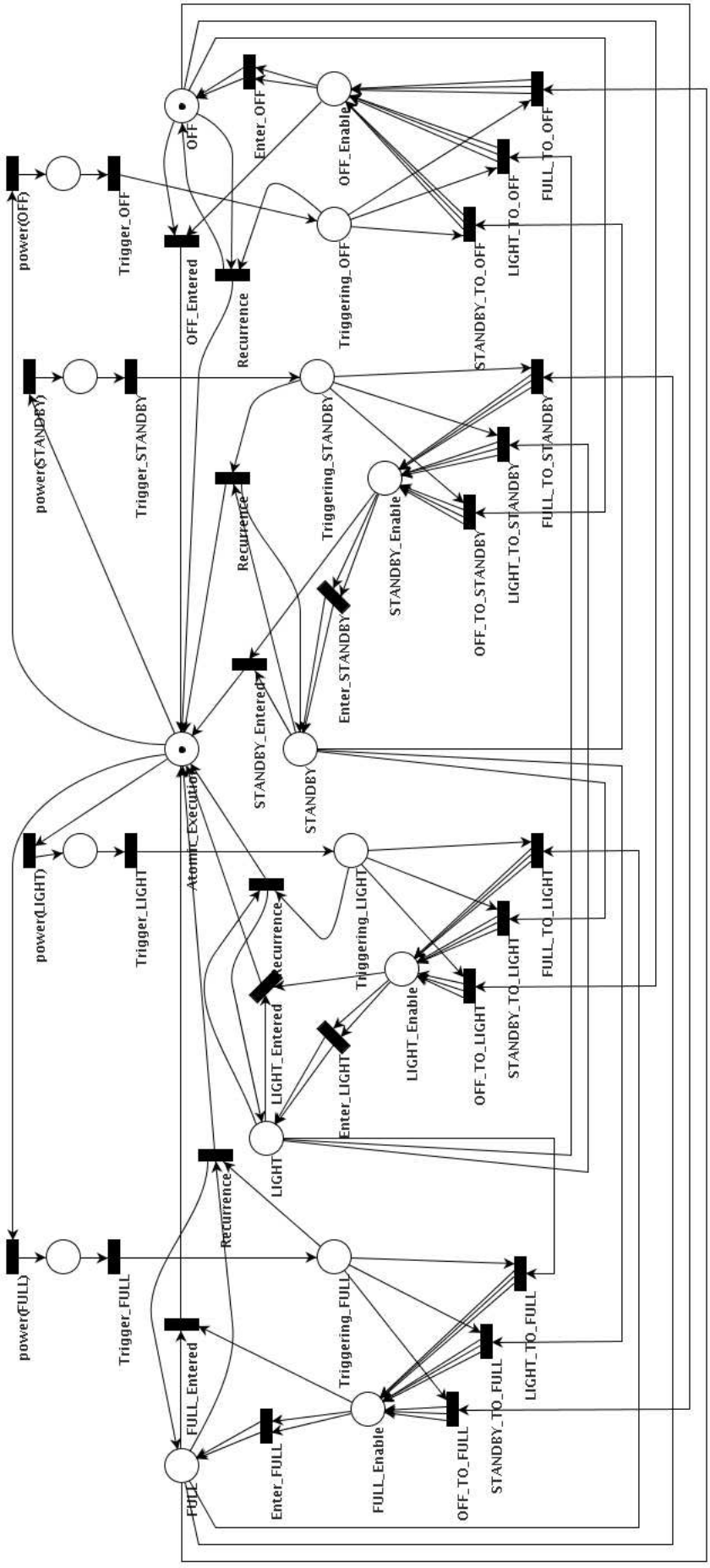
[WEI 02a] WEISSEL, A.; BELLOSA, F. Process cruise control: Event-driven clock scaling for dynamic power management. In: PROCEEDINGS OF THE INTERNATIONAL CONFERENCE ON COMPILERS, ARCHITECTURE, AND SYNTHESIS FOR EMBEDDED SYSTEMS CASES'02, 2002. **Proceedings...** Grenoble, France: [s.n.], 2002.

[WEI 02b] WEISSEL, A.; BEUTEL, B.; BELLOSA, F. Cooperative io - a novel io semantics for energy-aware applications. In: PROCEEDINGS OF THE FIFTH SYMPOSIUM ON OPERATING SYSTEMS DESIGN AND IMPLEMENTATION (OSDI '02), 2002. **Proceedings...** Boston, USA: [s.n.], 2002. p.117–129.

[ZEN 02] ZENG, H. et al. Ecosystem: managing energy as a first class operating system resource. In: ASPLOS-X: PROCEEDINGS OF THE 10TH INTERNATIONAL CONFERENCE ON ARCHITECTURAL SUPPORT FOR PROGRAMMING LANGUAGES AND OPERATING SYSTEMS, 2002. **Proceedings...** New York, NY, USA: ACM Press, 2002. p.123–132.

Apêndice A

Rede de Troca de Modos de Operação Gereralizada


power(OFF)
Trigger_OFF
OFF
Enter_OFF
OFF_Enable
FULL_TO_OFF
LIGHT_TO_OFF
STANDBY_TO_OFF
Triggering_OFF
Recurrence
OFF_Entered
power(STANDBY)
Trigger_STANDBY
Triggering_STANDBY
FULL_TO_STANDBY
LIGHT_TO_STANDBY
OFF_TO_STANDBY
STANDBY_Enable
Enter_STANDBY
STANDBY_Entered
STANDBY
power(LIGHT)
Trigger_LIGHT
Atomic_Execution
Triggering_LIGHT
FULL_TO_LIGHT
STANDBY_TO_LIGHT
OFF_TO_LIGHT
LIGHT_Enable
Enter_LIGHT
LIGHT_Entered
LIGHT
power(FULL)
Trigger_FULL
Triggering_FULL
LIGHT_TO_FULL
STANDBY_TO_FULL
OFF_TO_FULL
FULL_Enable
Enter_FULL
FULL_Entered
FULL